The Carpentries Curriculum Development Handbook

Erin Becker and François Michonneau

2022-08-29

Contents

Preface

This is work in progress. Comments and suggestions are welcome either as:

- an issue or a pull request on the GitHub repository
- or using hypothes.is

Chapter 1

Guiding principles

Carpentries curricula are developed based on the results of research in the science of teaching and learning. We rely on findings synthesized in the book How Learning Works: Seven Research-Based Principles for Smart Teaching (Ambrose et al., 2010). We use this text in our Instructor Trainer training program, so that Trainers (who teach our Instructor Training courses) can understand why we teach the way we do, and why our lessons are designed the way they are.

The authors identify seven principles of learning (direct quotation from the book are bolded):

1. **"Students' prior knowledge can help or hinder learning."** – Identifying what the learners know before coming to our workshops help us adjust what we teach. One way we do this is through our pre-workshop surveys. We also give the learners frequent exercises (or "challenges") throughout our lessons, which help Instructors and learners identify and correct misconceptions.
2. **"How students organize knowledge influences how they learn and apply what they know."** – Human working memory is limited, and can only handle about five to seven separate items of information at a time. We organize our lessons to introduce a few concepts at a time, and then provide challenges to give learners an opportunity to practice using these new concepts. This lets learners build connections between new concepts and their previous knowledge and transfer this new knowledge into their long-term memory, increasing the likelihood that they will be able to use this information successfully in new contexts.
3. **"Students' motivation determines, directs, and sustains what they do to learn."** – Our learners come to our workshops already motivated to learn the concepts we teach. They realize they need the skills we teach to conduct their research more effectively. They have experienced

the pain that comes with copying and pasting data across spreadsheets, or having to re-do complex graphs over and over as new data come in. However, they may also be intimidated by how much they have to learn before being proficient programmers and data analysts. Two strategies we use to keep our learners motivated are: (1) to create a positive learning environment, and (2) to teach the most useful skills first. We teach both of these in our Instructor Training program, and discuss how they influence curricular design below.

4. **"To develop mastery, students must acquire component skills, practice integrating them, and know when to apply what they have learned."** – Our lessons use frequent challenges to provide opportunities for learners to practice applying their new skills. These challenges are designed to incrementally build on each other and integrate previously taught and new skills. Careful attention to exercise design helps assure learners will be able to transfer the skills they acquire in our workshop to their own research.
5. **"Goal-directed practice coupled with targeted feedback enhances the quality of students' learning."** – When Learners try to solve the challenges we include in our lessons, they receive direct feedback from the computer - either an error message or the expected answer. Error messages are often opaque, and do not on their own help learners advance in their learning process, making them frustrated and demotivated. Our lessons are designed to be delivered as real-time in-person instruction, so that learners get feedback from Instructors and workshop helpers that is human-parsable and directed to their level of understanding. Furthermore, challenges used in a lesson should only require the skills that have already been introduced during the workshop, and should have a limited range of possible answers.
6. **"Students' current level of development interacts with the social, emotional, and intellectual climate of the course to impact learning."** – Providing a positive learning environment reduces learners' stress and helps increase their confidence in their ability to use the skills we teach. Creating this positive environment is a responsibility shared among all participants: Instructors, helpers, workshop hosts, and learners. Setting expectations by introducing our Code of Conduct at the start of each workshop, and enforcing it, contributes to making the workshop a welcoming space for everyone. Other strategies we use to create a positive learning environment are covered in our Instructor Training. Curricular content also plays a major role in creating a positive environment: examples chosen should not be alienating, skill level must be appropriate for the audience, and the examples and challenges must be directly applicable for our learners. For instance, when a learner creates a visualization that they can directly apply to their own data, it reinforces their motivation and favors a positive learning climate.
7. **"To become self-directed learners, students must learn to monitor and adjust their approaches to learning."** – In-person work-

shops allow Instructors to model the thinking process that is needed to address the challenges in our lessons. As an Instructor, being very explicit ("thinking aloud") about the steps of the mental model that are involved in identifying the functions to use, the values of the arguments they take, and the order in which to call these functions to solve an exercise, helps learners to think of the questions they need to ask themselves when facing new problems to solve. While this type of approach works for any level of complexity in the challenges we teach, it works best for the most advanced ones, where several steps need to be integrated to come to the solution. Before reaching this level of complexity, the challenges can be designed to guide this process, using scaffolding. Scaffolding is the process of providing support to a learner while new subjects and concepts are introduced. Scaffolding assists learners as they progress through increasingly complex code creation by breaking the complex code into smaller, more manageable chunks. Common practice problems used in instructional scaffolding are Parson's problems, where all the pieces of code to answer the problem are already written but are not in the correct order; and, fill in the blanks. Instructional scaffolding might be one of the most important things we use in our workshops. It sets learners on a successful path for further self-directed learning. When developing the content of the curriculum, think of the kind of thinking process that is needed to successfully address the research questions in your field.

Applying these principles effectively requires that they are incorporated into both **what** is taught (content) and **how** it is taught (delivery). Our Instructor Training program focuses on teaching Instructors how to use these principles in their teaching. In this handbook, we focus on applying these principles to curriculum design. Before starting to create lesson content, we highly recommend that you familiarize yourself with our Instructor Training curriculum.

1.1 Backward design

Backward design is an instructional design model that starts with identifying the desired outcomes of a learning experience, including core skills and concepts that learners need to acquire. These identified outcomes are used to develop course content and assessments to measure learners' progress towards these outcomes. This model was developed by Grant J. Wiggins and Jay McTighe in the late 1990s and is expanded on in their text Understanding by Design. We use backward design in developing our curricula because of its focus on identifying clear, measurable learning goals and providing assessments aligned with those goals.

In essense, the backward design process has three stages:

1. Identify the practical skills we aim to teach.

2. Design challenges to give an opportunity for our learners to practice and integrate these skills.
3. Identify what we need to teach for our learners to acquire these skills.

This approach ensures that all the skills we teach work together to meet the over-arching goals of our curriculum. It also reduces the risk that we won't teach a concept learners need in order to be able to master the skills we aim to teach. Similarly, it avoids teaching topics that do not help us (and our learners) meet our goals.

Reducing distractions is part of our lesson design as we strive to reduce cognitive load on learners. To this end, we also develop our lessons to be centered around a narrative and a dataset they can relate with quickly.

Because our workshops are domain-specific, the data we use, and the type of questions we ask with the data are already somewhat familiar to our learners. Their energy and focus can be directed towards learning the skills we teach rather than on getting familiar with data and concepts that are foreign to them. This strategy also increases the motivation of our learners. By learning how to solve problems that are familiar to them, they can more easily transpose these skills directly to their own data, and have a good starting point to continue their learning process as they try to solve new or more complex problems with their own data.

1.1.1 Identifying the practical skills

Our primary aim in a Carpentries workshop is to increase the confidence of our learners. We want to demystify and make accessible the process of computing and analyzing data. More than a third of learners at our workshops have little to no coding experience (Jordan et al., 2018). Our workshops provide them an opportunity to try, in a friendly environment, something they perceive as intimidating.

Another important goal is to make the research life of our learners easier. We emphasize teaching "good enough practices" (Wilson et al., 2017) - concrete skills that are accessible, able to be adopted by researchers of any skill level, and likely to make an immediate positive impact on learners' work. Teaching defensive programming, how to use spreadsheets effectively, or how to organise files consistently across research projects, are practical skills that can save a lot time when learners apply them in their own research.

When developing a new curriculum, the first step is to identify the skills that will be the most immediately useful to learners and have the biggest impact on their work. This will vary a lot, so having a clear idea of your lesson's intended audience is critical at this stage.

We will discuss in detail the process of defining your audience and identifying these core skills for your lesson in a later chapter.

1.1.2 Designing challenges to assess understanding

Once you have identified these high-impact skills, the lesson content should be designed to create frequent opportunities for learners to practice these skills while exemplifying the tasks they perform in their daily work. Live coding and hands-on challenges that learners can directly relate to should allow them to envision how they can start using the skills taught with their own data as soon as the workshop is over.

In traditional Western instruction, learners are presented with new material during course time and then sent home to practice applying the concepts learnt on their own. A major limitation of this approach is that learners often encounter difficulties in trying to apply their new knowledge or skills and need to troubleshoot on their own, without support. Education research shows that novices learn best when they are given feedback and coaching in real time while practicing their new skills (see principle number 5 above) so that errors are corrected and mis-steps redirected before mistakes have a chance to become discouraging or engrained in learners' memory.

To this end, Carpentries workshops are designed to provide frequent opportunities for learners to practice new skills. To be helpful in providing useful feedback, these challenges need to both be:

1. narrowly targeted to the skills that have been taught (i.e. not to depend on untaught concepts), and;
2. diagnostic (Instructors should be able to tell what the learner is misunderstanding based on how they answer the question).

The practical aspects of creating useful challenge problems is discussed in a later chapter.

1.1.3 Planning the content of the lesson

After deciding on a list of core skills your learners need, and creating a set of well-targeted, diagnostic exercises, you can then start to create the bulk of the content for the lesson. This material can be thought of as the "script" for the instructor to follow while teaching and should be planned very carefully to complement the exercises you've already designed. We cover the process of creating curricular content in a later chapter

1.2 Creating a narrative and selecting a dataset

Because we strive to provide a realistic experience for learners that is as similar as possible to the workflow they would use in their own work, our lessons use real data and are structured in a natural flow that corresponds to how a learner would experience working with their data in real life. For many curricula, this means starting with a lesson on data organisation and progressing through data cleaning, analysis, and visualisation or reporting. It is important to choose a dataset that is an authentic representation of what your audience would encounter in their day-to-day work. The practicalities of chosing an appropriate dataset are covered in a later section of this handbook.

1.3 Limitations of our approach

Learners can't go from complete novices to experts in two days (or in the course of any single class). We aim to provide learners with three things:

1. A set of foundational concepts and skills;
2. A mental model that connects those concepts into a useful framework and that can be built upon in their future learning; and
3. The motivation and skillset they need to continue learning past the end of the workshop.

Managing learners' expectations, and clearly communicating to them what they will (and won't) be able to do by the end of the workshop, is important because it limits the chance of demotivation.

Chapter 2

Our curriculum structure

Lesson Program	Lesson	Episode
Data Carpentry	Data Organization in Spreadsheets for Ecologists	Exporting data
	Data Wrangling and Processing for Genomics	Assessing Read Quality
Library Carpentry	Open Refine	Faceting and filtering
	The Unix Shell	Automating the tedious with loops
Software Carpentry	Plotting and Programming in Python	Writing Functions
	R para Análisis Científicos Reproducibles	Guardando datos

Data Carpentry lessons are domain-specific, and lessons that are taught together form a “curriculum”.

2.1 Episodes

An episode is a single block of content and renders as a single page of a lesson website. Each episode teaches a set of related concepts (for example, navigating files and directories or indexing and subsetting data). It lists measurable learning objectives related to those concepts and contains a number of challenges to assess learners' ability to perform those learning objectives. Episodes within a lesson can be dependent on earlier episodes and are generally taught in sequence.

2.2 Lessons

A lesson is a collection of episodes that together help the learner to develop a particular set of competencies (for example, version control or data organization). Each lesson has a landing page that lists all its episodes as well as the overall learning objectives. The overall learning objectives for the lesson should be met by its episodes. Assessment of lesson-level learning outcomes is summative and can be addressed with a post-workshop survey. Lessons can be used independently and should not rely on concepts from other lessons. Lessons may have optional episodes, but lesson designers should provide a recommended structure or structures for what episodes should be taught together and in what order.

2.3 Curricula

A curriculum is a set of lessons that together teach skills needed in a particular domain (for example, genomics or geospatial research). A curriculum has a landing page listing its component lessons and overall learning objectives, and describing the data used in the curriculum. Carpentries curricula often have a narrative structure, where lessons are explicitly linked in a sequence, for example, as steps in a data handling workflow.

2.4 Lesson Collections

A lesson collection is the full set of lessons housed within a specific Lesson Program (for example, all Software Carpentry lessons), or within The Carpentries organization (including our Instructor Training and Trainer Training lessons). Lessons within a collection should match the target audience of the Lesson Program or the broader Carpentries community.

Chapter 3

Deciding what to teach

As discussed in an earlier chapter, the first step in designing a curriculum according to backward design principles is to identify the practical skills that you aim to teach. This step is absolutely critical to defining the scope of your curriculum and to avoid scope creep, both during initial writing of the lesson materials and later community-driven development.

3.1 Target audience

To identify the skills you aim to teach, it is first essential to define your target audience, as different audiences will have different needs, as well as different starting skill sets. For example, Data Carpentry's Genomics workshop curriculum assumes that learners have some background in the biological sciences and will understand biological terminology and abbreviations used in those lessons, but does not assume any prior experience with the tools taught in the lesson, including The Unix Shell and Amazon Web Services. These assumptions set the stage for lesson development, by placing boundaries around what will and won't need to be explained in the lesson.

Defining your target audience is also essential to reducing the impact of expert awareness gaps. As an experienced researcher in your field, there are likely many steps in the data management and analysis process that you do without consciously thinking about. Without explicitly evaluating your target audience and understanding their actual background and skill level, you are in danger of skipping over intermediate steps that they need to know in order to succeed in their research.

You probably already have some sense of your target audience. To help refine this sense, ask yourself the following questions. Write down your answers and see if you can clearly articulate who is and isn't included in your audience. Share these thoughts with your colleagues and see if they agree.

3.1.1 Audience definition questions

- **What is the expected educational level of your audience?** – Do you expect most learners to be undergraduate students, graduate students, or to have completed graduate school? If you are targeting graduate students, do you expect learners to be new graduate students, masters degree holders, or doctoral candidates?
- **What type of exposure do your audience members have to the technologies you plan to teach?** – Think about the typical course work that someone in your field has completed when they are at your targeted educational level. Have they had classes where they needed to use R, Python, or some other programming language for their homework? Does your department require any courses on data organization or management? Do students in your field ever interact with a remote computing system? Don't worry if the answer to all of these questions is "no", most university departments don't build computational training into their undergraduate or graduate programs - which is why The Carpentries exists! Talk with others in your field, especially colleagues at different institutions and in different countries. Having an accurate picture of your target audience's actual exposure to these skills will help you plan a realistic curriculum.
- **What types of tools do they already use?** – Related to the previous question, it is useful to understand the toolkit that your target audience is already comfortable with. Do they commonly use spreadsheet software like Microsoft Excel, Numbers, or Google Sheets? Are they most comfortable working in rich text editors like Microsoft Word or Google Docs? Do they use any web-based GUIs or databases? Having this information will help you appropriately target your content by making useful analogies and tying new knowledge to existing knowledge. It is also very important to understand what tools established researchers in the field are using. No matter how enthusiastic a new doctoral student might be about using Python, if everyone else in their lab (including their advisor) uses MATLAB, they're unlikely to be successful in convincing the entire lab to change their workflows.
- **What are the pain points they are currently experiencing?** – The Carpentries trainings are designed to meet learners where they are and help them improve their workflows in a way that is immediately useful for them. We avoid idealism in favor of realism. Yes, it would be excellent if use of version control was standard across the research community, but if the learners at your workshop don't see the immediate benefit of version control, they are unlikely to implement it. Talk with students and other

new researchers in your field. What are the computational tasks they spend hours upon hours doing, only to have to redo when they get their reviews back from the publisher? What repetitive tasks do they do by hand and find mistakes in weeks or months later? People love to share stories like this and you can learn a lot about what others in your field are struggling with by collecting these stories. These are the skills you should be targeting in your lesson.

- **What types of data does your target audience work with? What are the commonalities in the datasets your target audience will encounter?** (types of variable, size, standard data formats, etc.) - If you're designing a domain-specific curriculum, you'll need to consider the range of data types that members of your domain community work with. For example, researchers in the social sciences work with a wide range of data types, but survey data is common in this research community. Data Carpentry elected to develop lessons around closed-ended survey data, with the hopes of expanding this lesson set to include analysis of free-response text in the future. Similarly, a genomics researcher may work with data sets that span multiple species or multiple individuals or populations within a species. The Data Carpentry Genomics curriculum development team chose to focus on intra-species data sets. Researchers in the field who work with different data types will still be able to benefit from the lessons, but choosing a common data type will ensure your lesson is maximally useful for a broad component of your domain community. It is important to make this decision early on, as trying to include every type of data that researchers in your field work with will result in sprawling, ungainly lessons that aren't useful for anyone. Choose one thing and do it well!

3.1.2 Learner profiles

After thinking through the audience definition questions above, and discussing these questions with colleagues in other institutions, you should have a fairly clear understanding of your target audience. You should now know who you expect to show up to your workshops, what knowledge and expectations they will bring with them, and what their motivations are. Keeping this information front-and-center throughout the lesson development process is incredibly important, as it is all too easy to forget your target learner and go down tangents in your lesson that don't serve this set of learner's needs. To make your target audience more concrete, we recommend creating a set of learner profiles. A learner profile describes a fictional learner at your workshop and includes the person's general background, the problems they face, and how the course will help them. Software Carpentry has example learner profiles that will be useful in developing learner profiles for your own course. We recommend creating 2-4 learner profiles that describe different segments of your target audience. These profiles can then be consulted at future stages in the curriculum development process. For example, when developing an exercise, you can look at your learner profiles and ask “Is this exercise useful for my target learners?”.

3.2 Skills list

Congratulations! You now have a solid understanding of the users of your lesson materials and can concretely define the background knowledge and goals of your learners. These goals are a combination of a) reducing or removing pain points that your learners can self-identify and b) achieving next-level competencies that learners may not realize are possible, but which will be practically useful to them in their research. You can think of these two components as the starting and ending points for your lesson. The background knowledge and skills that your learners bring to the workshop define the starting point, while your learners’ goals define the end point. With these start and end points, you can now define the list of skills that you will need to teach at your workshop.

3.3 Example using a Software Carpentry Learner Profile

The following example illustrates how a learner profile can be used to define a list of concrete skills.

> ### Example of Learner Profile
>
> Fan Fullerene is a graduate student in chemistry who is working as a lab technician to help cover his family’s living costs. His only programming experience is a general first-year introduction to computational science using Python.
>
> Fan’s supervisor is studying the production of fullerenes (also known as “buckyballs”). Each set of experiments involves testing a sample at 20 different temperatures and 15 different pressures. Using a machine borrowed from a collaborating lab, Fan can run all temperature and pressure combinations in one job, but must upload a parameter file to the machine to do this. The temperatures and pressures to be used vary from sample to sample, so Fan now has two dozen different parameter files, each containing 300 lines of control information that he fervently hopes is correct.
>
> The machine sends these files to Fan once the experiment is completed. Fan analyzes them by opening Excel, copying and pasting the data into a spreadsheet, then creating a chart using the chart

> wizard. He then saves the chart as a PNG file on the group's web site, along with the original data file.
>
> Fan and his wife have had two children arrive while in graduate school, and his research progress is behind that of his peers. He is very nervous about finishing his PhD and suffers from undiagnosed depression.
>
> Software Carpentry will teach Fan how to write programs to generate parameter files and analyze experimental results, and how to track the provenance of the data he is working with so that scientists can trace backward from the final charts to the raw data they represent. It will also teach him how to use version control systems to manage changes to his code.

Fan's background knowledge and skills include:

- maybe some vague recolection of basic syntax and terminology from his Python course
- use of Excel to process tabular data and create graphics
- interaction with a web GUI to upload data to his lab's website

Fan's goals include:

- creating parameter files automatically
- managing his many parameter files efficiently
- automating analysis for multiple runs of his experiment
- quickly and easily creating a specific graphic from his data
- uploading data and results to his lab's website

An (incomplete) list of target skills that can be extracted from this information includes:

- writing for loops (to create parameter lists)
- writing data to a file (e.g. parameter lists)
- writing reusable scripts
- creating a specific type of graphic programmatically
- customizing graphics to label them appropriately for a particular data set
- creating a version controlled repository for storing his parameter and output files
- pushing output files to his lab's website programatically

It is important at this stage to be sure you are defining **skills** that your learners will acquire, not **topics** that you will teach. You may be tempted to say, for example, that learners will learn about the grammar of graphics – which places the emphasis on a **topic** that learners will learn **about** - or that they will learn how to use the R package ggplot2 – which emphasizes the **tool** that learners will be exposed to. Neither of these ways of stating the learning goal focuses on the **abilities** that learners will develop that will help them in their work. Since you, as the lesson developer, will be using this skills list to create exercises and lesson content, it is essential to describe specific competencies and abilities that the content will help learners to develop. For example, this learning goal would be better phrased as "create plots that can be quickly and reproducibly modified and customized." If your target audience has specific plotting needs (like creating time-series plots, or visualising very large datasets), this learning goal would be phrased to incorporate those specific needs.

Spend as much time as you need to on defining this skills list. Don't rush it! This list will drive the rest of the curriculum development process. It's ok if your initial list needs to be modified later, but it's well worth the time to make this list as complete and concrete as possible now as it will save time in the remaining steps.

Chapter 4

Designing challenges

Now that you have a list of the concrete skills that your learners will develop in your workshop, it's time to start the second step of backward design - designing the challenges that you will use to help your learners practice those skills. These challenges will also enable the instructors to evaluate learner's skill progression in real time and re-direct their teaching as needed. Carpentries workshops use real-world data to increase the immediate applicability of our lessons and to reduce cognitive load for the learner. The first step towards developing the exercises you will use in your lessons is to select an appropriate dataset.

4.1 Picking a dataset

The dataset is a critical element of a Carpentries lesson. It needs to be chosen carefully and to meet the following criteria.

1. **Use a single dataset** – Curricula are domain-specific and the same dataset should be used across all lessons that are part of the same curriculum. When developing a standalone lesson (one that is not part of a curriculum), we encourage you to choose a dataset that is already in use in one of our existing curricula. Although each lesson should use the same dataset, it is often appropriate to use variations of the core dataset for different lessons within a curriculum. For instance, the Data Carpentry lessons on data organization with spreadsheets use messy spreadsheets that have been created from the original dataset, but which introduce formatting issues to teach tidy data principles. Whenever possible, these derived datasets should be created using scripts rather than manually, so they can be regenerated if the original dataset changes.
2. **The dataset should be released under a CC0 license** – Copyright laws and laws governing use and sharing of data and databases vary among

countries. The Creative Commons Zero (CC0) license is designed to allow unrestricted use and sharing of data universally. The CC0 license allows the development of lessons around the dataset and modification of the dataset to suit our teaching needs.

3. **The dataset should be deposited in a public repository** – All variations of the dataset that are used in the lesson should be deposited. The Carpentries deposits data for our lessons on figshare. If you choose another option, make sure the repository where the data is archived has the following features:

    - a DOI link pointing to an overview of the dataset
    - pre-registration of the DOI
    - all files can be downloaded directly as an archive (e.g., zip file) with a persistent link
    - each file can be downloaded directly with a persistent link
    - the repository supports versioning

4. **The dataset should be real and represent what researchers in the field encounter** – The datasets used as examples in the lessons should be based on real research datasets, and be of sufficient complexity that they are representative of the type of dataset that learners would encounter in their own research.

5. **Authors of the dataset should be identifiable, acknowledged, and there should be a link to the original source for the data** – Even though the datasets we use in our lessons are released under a CC0 license, we acknowledge the authors of the dataset and link to the research projects based on the data we use.

6. **The dataset should be large enough** – Analysing the dataset should represent a real challenge that highlights the power and usefulness of the tools covered in the lessons. The dataset should be larger than what would be easy to analyze and manipulate in a spreadsheet program. It should be similar in size to what researchers in that domain work with in their actual research. For instance, the core dataset for the Data Carpentry Ecology curriculum has ~35,000 rows.

7. **The dataset should be complex enough to ask interesting questions** – Each observation should have at least 4-5 variables. These variables should be of a few different data types (at least continuous, discrete, integers, real numbers; and depending on the domain, may include more specialized data types such as date/time, GPS coordinates, unstructured text, etc.)

8. **The motivation for study and the protocol for data collection should be understandable without much context** – We have limited time in our workshops to cover the technical skills we want to teach. It

should not take long to explain to learners what the data is about, how it was collected, and what types of interesting questions can be asked from it.

9. **The dataset should be relevant in different geographical and cultural contexts** – Our workshops are taught to learners from diverse cultural and geographical backgrounds. The dataset should be understandable without much cultural context or pre-requisite knowledge needed to make it compelling.

10. **There should be clear and comprehensive metadata** – The metadata should include a description of the data, explain what is included in each data field, how it was measured, and the unit in which it is reported.

Overall, datasets used in Carpentries workshops should serve as examples of publicly deposited data suitable for research re-use. Learners should be able to use these datasets as examples and guides for their own research data that they would like to publish and make available to the broad scientific and academic community.

4.2 Formatting the dataset for teaching

A possible challenge when using research datasets for teaching is that the dataset can include complexity that makes teaching more difficult by unnecessarily increasing learners' cognitive load. While it is important for the dataset to provide an authentic experience for learners, it is often useful to simplify it or do some initial data cleaning and wrangling to make it easier for learners to focus on the core skills you are teaching. For instance, you may want to edit the dataset so that missing values are parsed appropriately. You may also want to remove data which leads to errors or warnings during parsing, columns with data types that are not relevant for the learning objectives of the workshops, or variables which require additional context to understand.

When preparing a dataset for teaching, aim to find a balance between providing an authentic experience for learners while keeping complexity low to limit distractions from the learning objectives. Depending on the lesson's goals, it might also be interesting to include several versions of the datasets that have undergone various levels of processing. At the beginning of the lesson, you can provide a clean and well organized dataset, while later you can introduce more complexity and teach learners how to handle it to generate the cleaner version of the data. Don't introduce too many (no more than three) versions of the dataset in your lessons, as dealing with many files and remembering their differences can become challenging for the learners.

4.3 Designing challenges

Once the dataset is in place, you can start to design challenges that provide learners an opportunity to practice the skills that you've included in your skills list. Writing the challenges before writing the content of the lesson ensures that the lesson will remain focused, and can reveal gaps in your skills list.

The challenges in a lesson should be a mixture of direct application challenges and synthesis challenges. A direct application challenge is a straightforward implementation of a concept that learners have just been exposed to, while a synthesis challenge requires learners to integrate recently learned skills with skills that were covered earlier in the lesson. Learning is reinforced when Instructors explicitly point out how the skills seen in earlier parts of the lesson are being integrated to solve the challenges.

Challenges in Carpentries lessons are a form of formative assessment. They help learners further their learning by having a chance to put into practice the skills being taught. They also help Instructors monitor the level of understanding in the classroom, and potentially catch misconceptions in the learner's mental models that can be corrected in real time, before they become ingrained.

When starting to design challenges, it is helpful to start by planning the last exercise of each episode. This will help you keep the big picture in mind and ensure that the rest of the exercises you develop lead up to this larger goal. These final challenges are also the most likely to be "unscaffolded", and so are easier to develop without detailed knowledge of the various types of exercises discussed later in this chapter. So you can go ahead and draft those final challenges now before reading the rest!

4.4 Different types of challenges

There are many different types of challenge questions that can be developed. In this section, we introduce a few common types of challenges, in order of increasing difficulty (for the learner). When planning your exercises, keep in mind that you will need at least one exercise for every 10-15 minutes of instruction (and more is better!). To quickly estimate how many exercises you need to develop, divide the number of minutes of total delivery time (the number of minutes your learners will be in their seats) by 20. For example, a two hour lesson needs around six exercises. As a rough estimate, plan 8 minutes for each exercise (including discussion of the solution and questions) and 12 minutes of instructional time between exercises.

4.4.1 Multiple Choice Questions

Multiple choice questions (MCQs) can be a useful tool for formative assessment if they are designed such that each incorrect answer helps the Instructor to identify learners' misconceptions. Each incorrect answer should be a *plausible distractor* with *diagnostic power*. "Plausible" means that an answer looks like it could be right, and "diagnostic power" means that each of the distractors helps the instructor figure out what concepts learners are having difficulty with.

For example, if learners are learning about subsetting data in `R` with the `dplyr` functions `filter()` and `select()`, you might ask them, using the `palmerpenguins` dataset to determine which of the following code blocks will return only the values of the "species" and "body_mass_g" variables for observations from the Dream island collected after 2007.

a.

```r
penguins %>%
  filter(island == "Dream" & year > 2007) %>%
  select(species, body_mass_g)
```

This is the correct answer.

b.

```r
penguins %>%
  select(island == "Dream" & year > 2007) %>%
  filter(species, body_mass_g)
```

Learners who select this answer have a factual misconception. They have confused the `select()` and `filter()` functions. They need to be reminded that `filter()` is for subsetting by row and `select()` is for subsetting by column.

c.

```r
penguins %>%
  select(species, body_mass_g) %>%
  filter(island == "Dream" & year > 2007)
```

Learners who select this answer have a conceptual misunderstanding that may require more time and effort to correct. They haven't understood that the pipe (`%>%`) character only passes into the next command the output of the previous command, and **nothing else**. Since they have used the `select()` function first, the “island” column is no longer present in the output and cannot be used for comparison. This misconception might be addressed by drawing a diagram and walking through what the data looks like at each step of the command. Another follow-up question using the same skills could then be used to assess whether learners have understood the concept.

As illustrated above, formative assessments are most powerful when an instructor modifies their instruction depending on the results of the assessment. An instructor may learn they need to change their pace or review a particular concept. Knowing how to respond to the results of a formative assessment is a skill that you will develop over time. Making sure your assessments are designed to test only one or two concepts at a time will help ensure that the feedback is useful.

The process of developing diagnostic plausible distractors takes time and requires some knowledge of what common misconceptions are for a particular topic. This knowledge can come through teaching experience (yourself or others’) and is sometimes formally defined through concept inventories. One strength of The Carpentries community is that our lessons are taught over and over again by different Instructors in different teaching contexts. Some of those Instructors give feedback on challenges and misconceptions that their learners had. Our exercises are thus continuously improved by pooling the teaching experience of our 1,500+ strong Instructor community!

4.4.2 Parson’s problems

One reason well-designed multiple choice questions are so useful is that they constrain the problem space. Learners don’t need to worry about all of the details of syntax and how to spell all of the variable names, but can focus on just the concepts that the exercise author intended them to focus on. Another type of formative assessment that provides this benefit are Parson’s problems. A Parson’s problem is an exercise where learners are given a set of items (in our case, lines of code) and asked to put them into an appropriate order to accomplish a specific task. The MCQ example given above could be formulated as a Parson’s problem:

```
filter(island == "Dream" & year > 2007) %>%
penguins %>%
select(species, body_mass_g)
```

A more difficult version of a Parson’s problem might include lines that are not part of the solution (*distractors*):

```
filter(island == "Dream" & year > 2007) %>%
select(island == "Dream" & year > 2007) %>%
penguins %>%
filter(species, body_mass_g)
select(species, body_mass_g)
```

If this is the case, make sure learners know that not all of the code chunks need to be included in their answer!

Parson's problems are somewhat less structured than MCQs, which makes them slightly better for preparing learners to tackle similar problems in their own work. However, this also makes it more difficult for Instructors to diagnose learner misconceptions and adjust their teaching accordingly (because there are more possible responses). As will all of the challenge types we will discuss in this chapter, MCQs and Parson's problems can be used in combination to provide learners with both structure and appropriate levels of challenge.

4.4.3 Fill-in-the-blank problems

Fill-in-the blank problems can be thought of as the next level of decreasing structure after MCQs and Parson's problems (although this depends on the number of lines in the Parson's problem and the number of possible choices for filling in the blanks, among other factors). The following challenge (from the Software Carpentry lesson on The Unix Shell) illustrates one possible application of fill-in-the-blank problems:

Challenge: Moving to the Current Folder

After running the following commands, Jamie realizes that she put the files `sucrose.dat` and `maltose.dat` into the wrong folder:

```
$ ls -F
 analyzed/ raw/
$ ls -F analyzed
fructose.dat glucose.dat maltose.dat sucrose.dat
$ cd raw/
```

Fill in the blanks to move these files to the current folder (i.e., the one she is currently in):

```
$ mv ___/sucrose.dat  ___/maltose.dat ___
```

Solution

> ```
> $ mv ../analyzed/sucrose.dat ../analyzed/maltose.dat .
> ```
>
> Recall that `..` refers to the parent directory (i.e. one above the current directory) and that `.` refers to the current directory.

We can also apply this concept to our earlier example and ask learners to fill-in-the-blanks to build a code block that will return only the values of the “species” and “body_mass_g” variables and only for observations from the “Dream” island collected after 2007:

```
penguins %>%
  ________(island __ "Dream" ___ year > 2007) %>%
  ________(species, ___ )
```

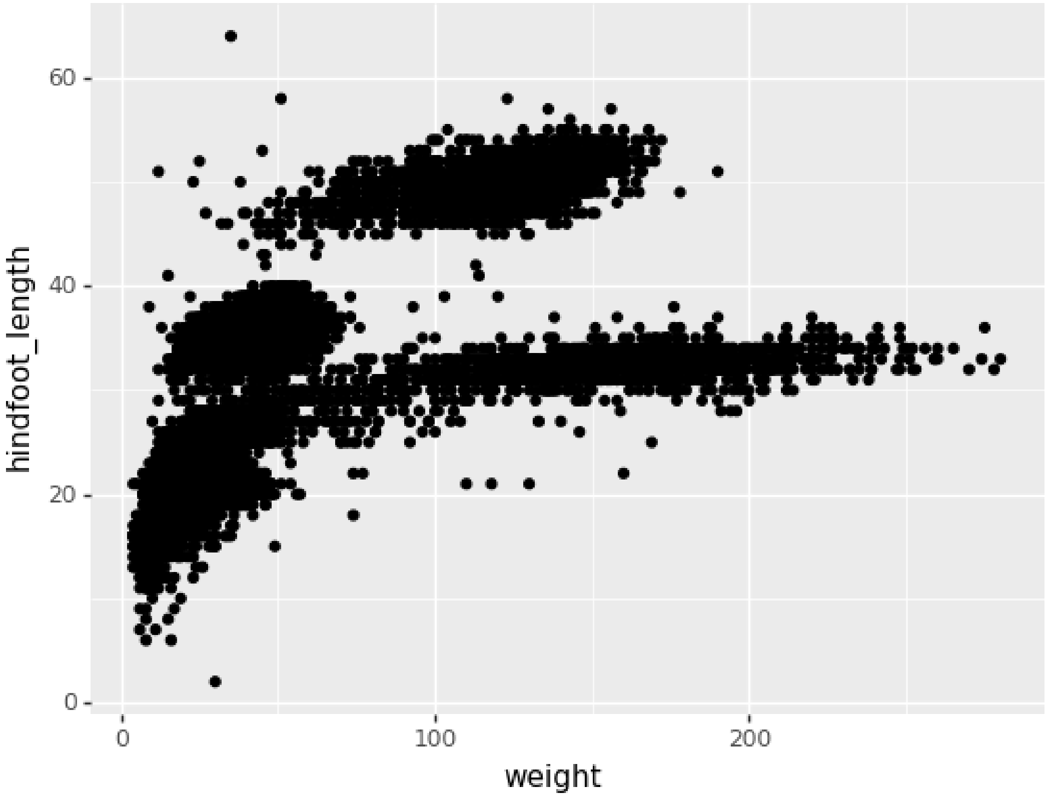
The difficulty of a fill-in-the-blank problem can be adjusted by changing the number of blanks, and by providing (or not providing) a “word bank” of options to use. You can even use a series of similar fill-in-the-blank problems, increasing the number of blanks each time, to prepare learners to build a code chunk without scaffolding. When used in this way, fill-in-the-blank problems are also called *faded examples*. We discuss the use of faded examples, and why they are a useful tool, in more detail in our Instructor Training course.

4.4.4 Use the concept in a different context

Once learners have had an opportunity to practice using a concept in the same context as it was originally taught (direct application challenges), it’s time to stretch their understanding by asking them to apply the same concept in a different context. This adds realism and makes learners better prepared to apply these concepts to unique problems in their own work. This type of exercise will often require learners to proactively look up help files or do a Google search.

For example, in the Data Carpentry lesson Data Analysis and Visualization in Python for Ecologists, plotting is taught using the `plotnine` package. That package implements the grammar of graphics developed by Leland Wilkinson, which includes the concept of a plot’s *geometry*. Different plot types have different underlying geometries that interact differently with other plot parameters. In this lesson, learners are led through an example where they create a scatterplot (using `geom_point()`) of weight versus an animal’s hindfoot length.

```
surveys_plot = p9.ggplot(data=surveys_complete,
                         mapping=p9.aes(x='weight', y='hindfoot_length'))
surveys_plot + p9.geom_point()
```



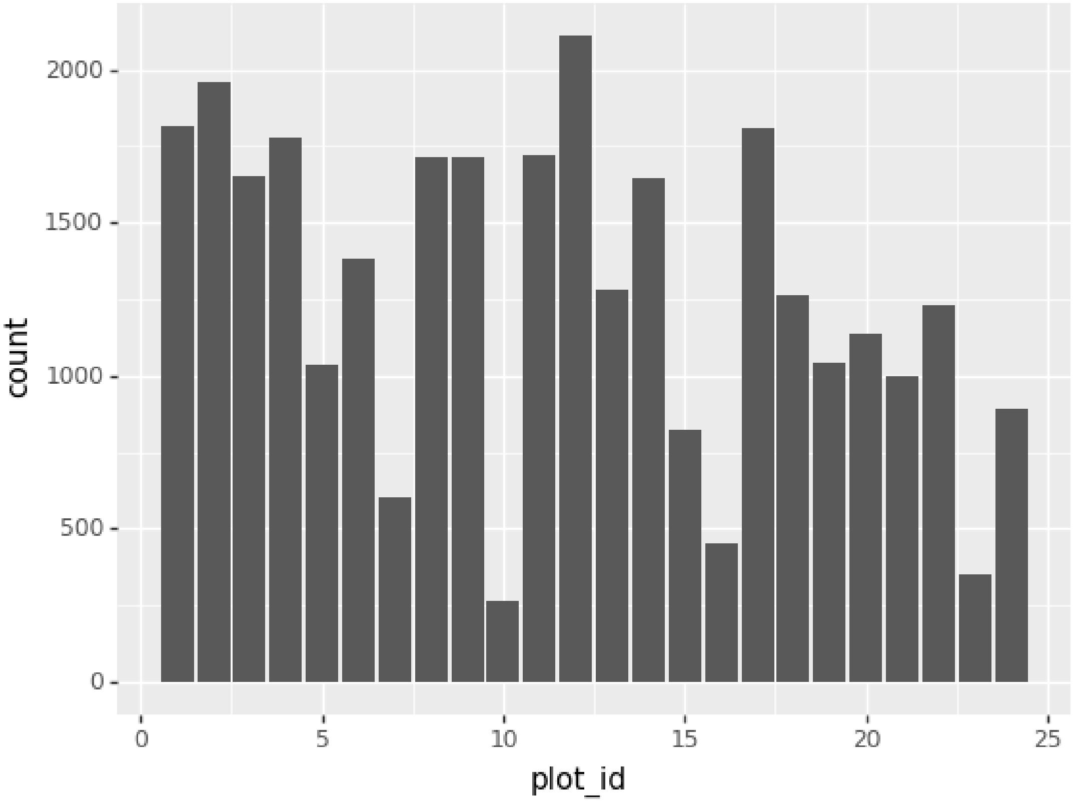
They are then asked to create a barchart showing the number of observations in each region (or "plot") of the field site.

Challenge - bar chart

Working on the `surveys_complete` data set, use the `plot-id` column to create a `bar`-plot that counts the number of records for each plot. (Check the documentation of the bar geometry to handle the counts)

Answers

```
(p9.ggplot(data=surveys_complete,
           mapping=p9.aes(x='plot_id'))
    + p9.geom_bar()
)
```



To answer this problem, learners need to locate and decipher the appropriate help file (for the `geom_bar()` function). They also need to change the `mapping` attribute to apply to "plot_id" rather than "weight" and "hindfoot_length". Later challenges in this lesson require learners to apply multiple concepts in new contexts, for example, modifying both the plot type and underlying *aesthetics* like color.

These types of problems are incredibly powerful, as they provide the most realistic example we've discussed so far of the skills learners will need to be able to apply what they've learned to their own work. This also means, however, that these problems are fairly unstructured, and include many more potential areas for learners to get off track. It is also very easy, when designing these problems, to make them more difficult than you intended - either by requiring learners to apply multiple concepts in new contexts or by including new concepts that haven't yet been covered in the lesson. Be careful, always have your lesson reviewed by at least one novice user if possible, and above all else, be prepared for feedback from Instructors "in the field" about how to improve these (and all of your) challenge problems.

Chapter 5

Developing content

At this point in the lesson development process, you should have a list of the core skills that your lesson will teach, along with a set of exercises that you will incorporate into your lesson. If you've taken the time to go through these steps, the remaining parts of the development process will be much more straightforward and take less time than if you try to start with developing content. Its ok if your list of skills and set of exercises changes a little as you develop your lesson content – none of us get it right on the first try – however, if you've invested time in the first two steps of the backward design process, these changes should be fairly small.

Now that we're ready to start writing the lesson material, it's worth laying out some of the fundamental elements of a Carpentries lesson. These elements are covered in much more detail in a later chapter, so we will focus here on the bare minimum you need to know to get started. It's perfectly ok to work through some or all of the steps described in this chapter before worrying about the logistics of putting your materials up on GitHub, but if you're working with multiple authors, it may be worth putting your materials on GitHub at this stage, to make collaboration easier. The Technological Introductions chapter can get you started with hosting your lesson on GitHub.

5.1 Fundamental elements

The bulk of a Carpentries lesson consists of exercises, example code chunks, and narrative text. You have already developed many or all of the exercises you will include in your lesson, so congratulations on being one-third done with content development! The remaining two major components of the lesson are described below.

5.1.1 Code chunks

Carpentries workshops are taught using participatory live coding. Instructors type the code as they teach it and learners type along with the Instructor. For more information about how live coding works, and what its advantages and disadvantages are, read that section of our Instructor Training program. The fact that Carpentries workshops are taught using live coding means that much of your episode content will be *code chunks* - short blocks of code that learners type along with the Instructor and evaluate on their own machines.

Since you already know the structure of your dataset, and already have your exercises in place, it's fairly straightforward to create the code chunks for your lesson. Starting with the first exercise, make a list of each of the commands and syntax elements that learners need to be familiar with in order to solve the exercise. For example, the first exercise in the Data Carpentry lesson Introduction to the Command Line for Genomics asks learners to:

> ## Challenge
>
> Use the `-l` option for the `ls` command to display more information for each item in the directory. What is one piece of additional information this long format gives you that you don't see with the bare `ls` command?

In order to solve this challenge, learners need to know the following (in approximately reverse order):

1. That options come after the command (e.g. `ls -l` not `-l ls`).
2. How to find the manual page for `ls` to understand the output.
3. What the `ls` command does in its bare form.
4. How to navigate to the right directory.
5. How to tell what directory they're in.
6. How to open the bash shell on their machines.

That's *six* concepts or pieces of information that learners need to have to answer this challenge! If it didn't seem that complex to you, please revisit the idea of expert awareness gap in our Instructor Training curriculum.

With this list, you can now construct the pieces of code that an Instructor will need to walk learners through before this first exercise. Those pieces of code (in the same order as above - the reverse order they would be used in a lesson), are something like:

1. `ls -F` (or any other commonly used `ls` option other than `ls`)
2. `man ls` (or `man cd` or `man pwd`)

3. `ls`
4. `cd`
5. `pwd`
6. Not a code chunk, but a demonstration of how to open the bash shell on the Instructor's machine.

You now have your first set of code chunks corresponding with your first exercise. Put these in reverse order and you're well on your way to writing your lesson!

5.1.2 Narrative text

The narrative text component of the lesson provides a guide for Instructors to use while teaching. It should provide a clear and complete narrative that (in theory) could be used by an Instructor as a script for the lesson. Instructors *will not* actually be using this text as a script, because they will be responding to the needs of their particular learners in real time, both in terms of the level of detail they go into, and in how they answer questions. However, this text should give Instructors a starting point for explaining the lesson content. This text can also be used by independent learners to work through the lesson materials outside of a workshop setting.

Wait to start writing your narrative text until you're fairly confident about the set of code chunks you've written. Any changes to the code chunks will require corresponding changes to the text, which can become time consuming. For the set of code chunks above, we would need to write a short explanation for each step. For example, the narrative corresponding to the `ls` code chunk in this lesson is:

We can see files and subdirectories are in this directory by running `ls`, which stands for "listing":

```
ls
```

`ls` prints the names of the files and directories in the current directory in alphabetical order, arranged neatly into columns.

And the narrative corresponding to the `man ls` command in this lesson is:

`ls` has lots of other options. To find out what they are, we can type:

```
man ls
```

Some manual files are very long. You can scroll through the file using your keyboard's down arrow or use the Space key to go forward one page and the b key to go backwards one page. When you are done reading, hit q to quit.

Narrative text should be as short as possible and should avoid discussion of edge cases or caveats. If there are important caveats, those can be included in a callout box (see the Technological Introductions chapter for formatting details of callout boxes). Only include callouts for cases a significant fraction of your learners will experience in their work. It's very easy to clutter the lesson and overwhelm learners.

5.1.3 Considering cognitive load

You may have noticed that the exercise we dissected above required six pieces of knowledge to solve. This number isn't random! Cognitive science research has clearly established that human working memory is limited, and can only handle a small number of separate items of information at once. This number is generally considered to be 7 plus or minus 2, but recent research has suggested it may be even smaller. If learners are presented with more than this many discrete pieces of information, they will not be able to keep all those pieces in mind simultaneously. Once they have had an opportunity to work with those pieces of information, their brains will create connections that will allow them to store the information in long-term memory and retrieve it as needed. This is why it is essential to include frequent exercises, spread throughout the lesson at regular intervals with only ~5 intervening new concepts. If you find that you need more than about this many code chunks between exercises, you probably need to either simplify your exercise, or add another exercise in between.

5.2 Supporting elements

Once you've created your exercises and written your code chunks and narrative, your lesson is nearly complete! There are a few remaining elements that support the main episode content by providing a clear outline of content for both learners and Instructors. These supporting elements are learning objectives, key points, and framing questions. When you create your lesson repository from The Carpentries template, there will be sections for you to enter these supporting items for each episode. The template will extract these elements and embed them in the rendered lesson webpage. For details, see the Technological Introductions chapter.

5.2.1 Learning objectives

Great news - you've already almost completed writing your learning objectives! The skills list that you developed for your lesson can be easily transformed to learning objectives. Learning objectives are statements that communicate to learners the skills they can expect to gain from the lesson. They should always be framed from the learner's perspective and use action words. In other words, they should emphasize what a learner will be able to *do* not what they will *know*. For example, the first episode of the Data Carpentry lesson Introduction to the Command Line for Genomics includes the following learning objectives:

- Describe key reasons for learning shell.
- Navigate your file system using the command line.
- Access and read help files for bash programs and use help files to identify useful command options.
- Demonstrate the use of tab completion, and explain its advantages.

If you didn't use action words when creating your skills list, there are many existing resources available that list action verbs associated with different levels of learning (one example). For our purposes, the differences among these levels isn't as important as using action verbs in defining your learning objectives. When learning objectives are framed in this way, learners should be able to self-evaluate whether they have completed each learning objective and concretely understand what they have gained from the lesson or what they still need to work on.

Keeping in mind our discussion of cognitive load above, each episode should have ~5-7 learning objectives. If you have more than that, you should consider splitting the material into multiple episodes.

5.2.2 Key points

While learning objectives communicate to learners the skills they should develop by working through each episode, key points summarize the main pieces of knowledge that learners should remember after completing the episode. These should also be limited to ~5-7 items. For the episode we've been considering in the Data Carpentry lesson Introduction to the Command Line for Genomics, the key points are:

- "The shell gives you the ability to work more efficiently by using keyboard commands rather than a GUI."
- "Useful commands for navigating your file system include: `ls`, `pwd`, and `cd`."
- "Most commands take options (flags) which begin with a `-`."
- "Tab completion can reduce errors from mistyping and make work more efficient in the shell."

The Carpentries lesson template automatically creates a reference page that includes all of the key points for each episode in the lesson. Key points should be specific enough that learners are able to use this reference page as review.

5.2.3 Framing questions

The final supporting component for the lesson are the framing questions for each episode. These questions provide a high-level overview of the motivations for learning the lesson content. There should be ~1-3 questions for each episode, which should correspond to questions that your target audience ask themselves about their own workflows. The set of framing questions for the Data Carpentry lesson Introduction to the Command Line for Genomics includes:

- How can I view and search file contents?
- How can I create, copy and delete files and directories?
- How can I control who has permission to modify a file?
- How can I repeat recently used commands?

The lesson template automatically creates an overview page for the lesson, which includes the framing questions for each episode.

Chapter 6

Community development roles

Our lessons are intended to be teachable by any certified Carpentries Instructor with the appropriate domain experience and background knowledge. Lessons should also be appropriate for learners at different institutions, and not require specialized local or institutional knowledge. Our lessons attain this broad usability by virtue of being the product of many people at different institutions around the world who work together in different roles to create, test, and iteratively improve and update lesson materials.

6.1 Lesson Authors

A lesson may have one or several initial authors. Authors draft the lesson content, figures, and code and create appropriate challenge problems. Authors should have both appropriate domain experience - working in the same field as the intended audience for the materials, and programmatic experience – regularly using the tools for which they are developing lessons in their own work. From a technical standpoint, authors will also need to be familiar with specific technologies that we use for developing and hosting The Carpentries lessons - including git, GitHub, Markdown, RMarkdown, … But don't panic! If you're not comfortable with any or all of these tools, we'll walk you through what you need to know to use them later in this Handbook. Most importantly, to ensure that lesson materials are consistent with The Carpentries vision and values, lesson authors should be certified Carpentries Instructors or otherwise involved in The Carpentries community. If a group of authors are writing a lesson collaboratively, we recommend dividing lessons up by episode, and having only one author per episode. Clearly defining the learning objectives for each episode will help avoid overlap and ensure the lesson flows smoothly - but it will still be important to have regular checkins with all authors.

6.2 Reviewers

No one is perfect! Lesson materials should be read and tested by at least one person other than the original author before being released for use by the broader community in beta pilot workshops. If a lesson has more than one author, co-authors can review each other's content. Community members can also serve as reviewers. A reviewer commits to carefully reading and testing all code for a lesson or set of episodes and leaving detailed feedback for lesson authors to correct any errors or other issues found. This feedback is provided as issues and/or pull requests (PRs) in the lesson's GitHub repository (we'll explain all of that soon!). Authors modify the lessons based on reviewer feedback to ensure the lesson is bug-free, all code runs as expected, exercises are appropriate and test only the concepts being taught, and in general that the lesson is ready to be delivered to learners in a beta pilot workshop.

Reviewers will also check to be sure the lesson complies with our our accessibility requirements.

Don't worry - this review stage is not the only point at which lessons will receive feedback. Our collaborative lesson development model ensures that Instructors and other community members will continue to engage with the lesson materials at all stages of their development and provide near real-time feedback to keep the lessons in good shape for as long as they are actively being taught!

Reviewers do not need to have any particular domain background or tool expertise. In fact, we recommend including reviewers who are complete novices either the tool being taught by the lesson and/or the lesson's target domain. This can help overcome authors' expert awareness gap.

Reviewers should also include people from different geographic regions and cultural and linguistic contexts than the lessons' authors. If all of a lesson's authors are based in the UK, for example, there should be at least one reviewer from outside the UK, and ideally from a non-majority English-speaking area. Ensuring that lessons are reviewed by people from a variety of cultural and linguistic contexts helps us to avoid colloquialisms, culturally-specific references, and other issues that might make our lessons less accessible to a global community. You will likely need one or two reviewers for every two hours of lesson content. A four-hour (half day) lesson should have at least two to four reviewers.

6.3 Lesson Maintainers

Lesson Maintainers are essential for the long-term viability of a lesson. As a lesson is taught, new Instructors and learners identify potential places for improvement - whether correcting a typo, simplifying code, or suggesting a significant shift in the narrative of a lesson. Maintainers proactively monitor their lesson's GitHub repository to make sure that PRs and issues are addressed in a timely manner. Maintainers also play a vital role in communicating with contributors, ensuring that our community lives up to its ideals in welcoming and appreciating contributions from everyone - from first-time contributors to long-time members of The Carpentries community.

People acting as Maintainers should be experienced with the tool that is being taught in the lesson, ideally using it daily or weekly in their own work. In addition, they should have experience working in a relevant domain related to the lesson materials and/or experience working with GitHub and the other technologies we use to create and host our lessons. Each lesson will have at least two Maintainers, and it's ok for one Maintainer to have domain experience and another to be more comfortable with the technical aspects of lesson maintenance.

6.3.1 Maintainer recruitment, requirements and time commitment

It's a good idea to recruit at least four Maintainers per lesson, as our community members are volunteers and may have fluctuating levels of time to commit to this role. Having a larger team of Maintainers ensures you'll always have at least one or two active Maintainers at any given time.

Maintainers commit to a specific set of duties, and go through a short onboarding process to familiarise them with the social, technological, and curriculum-specific elements of lesson maintenance. This onboarding is available as a written curriculum and in future may be made available as a video recording, however, if time allows, we recommend running onboarding for your Maintainer group via synchronous video-conference. Getting everyone in the same virtual room for these discussions creates a sense of community and will make it easier for your lesson Maintainers to work together. Information about how to run a synchronous Maintainer onboarding, including email templates and scheduling information, is available in The Carpentries Handbook.

Once your Maintainers have been onboarded, they will join The Carpentries full Maintainer team. This team meets monthly to share skills, exchange ideas, and get suggestions from each other around challenging issues and pull requests in their individual lesson repositories. The minutes for these meetings are stored in the Maintainer Resources repository. Meeting times and information about how to stay in touch with the Maintainer team is available in The Carpentries Handbook.

A lesson should have at least three onboarded Maintainers before it enters the beta pilot workshop phase. Before that time, lesson feedback and edits will generally be managed by the lesson authors.

6.4 Curriculum Advisors

Curriculum Advisors provide high-level oversight, vision, and leadership for a curriculum and guide large-scale updates. Unlike Maintainers, who are responsible for the day-to-day work of keeping lessons stable and teachable, Curriculum Advisors maintain a broader perspective on the state of the field and make strategic decisions about major changes to a lesson, for example, updating the technology being taught to take into account major advances in the field or changing the dataset used in the lessons to appeal to a broader group of learners.

A Curriculum Advisory Committee (CAC) is composed of 5-8 people with significant domain expertise who represent the breadth of the field that a curriculum is intended to reach. For example, the Data Carpentry Geospatial CAC includes researchers in ecology, limnology, environmental sciences, and sociology, along with university staff leading institutional GIS education efforts. Multiple career levels are represented, from PhD candidates at the end of their graduate work, to mid-late career professionals. At least one member of the CAC should be actively teaching in the field, so they can bring a practical perspective about what skills students and early-career researchers need. Lesson Maintainers may also serve as Curriculum Advisors, but most Curriculum Advisors will not also be Maintainers. Curriculum Advisors commit to a minimum one-year term, but may serve multiple terms. A CAC should include members from multiple geographic regions and cultural and linguistic contexts to ensure that the curriculum meets the needs of our global community.

A Curriculum Advisory Committee meets virtually approximately twice a year to discuss and make decisions about proposed large-scale changes to the lessons within their curriculum. These proposals may be initiated by community members, including Maintainers, or by members of the CAC. The CAC communicates their recommendations back to the Lesson Maintainers and provides consulting and support to Maintainers in implementing proposed changes.

Ideally, a Curriculum Advisory Committee should be assembled in the initial stages of lesson development, before materials start to be drafted. The CAC can then provide high-level guidance, including defining the learning objectives and core content for the curriculum and selecting an appropriate dataset that will speak to a broad group of learners. However, we recognize that lesson development may be part of a grant or other structure with requirements that are incompatible with putting together a CAC at such an early stage. If a curriculum will be included in the official Carpentries lesson stack, there must be a Curriculum Advisory Committee in place at the time of its first publication. The CAC should meet regularly for as long as a curriculum remains active.

6.5 Beta Pilot Instructors

A new lesson or curriculum is often taught for the first time locally at the organization that houses the lesson authors. This can be an opportunity to troubleshoot organizational or technological issues with the lesson, and should certainly be done if resources allow, but is not a sufficient test of the lesson's broader teachability. It is always easier for the creator of a set of curricular materials to use those materials, but more difficult to communicate all of the relevant details to enable others to deliver the curriculum efficiently. In order to ensure that our lessons are able to be taught by all appropriately experienced certified Instructors, a lesson should be beta piloted at least twice outside of the institution in which it was developed, ideally in two different countries. Because lessons at this stage are expected to still have some technical and flow issues, instructors for these beta pilot workshops should be certified Carpentries Instructors who have previously taught at least two Carpentries workshops. Instructors with this level of experience will be more prepared to troubleshoot issues that arise during the workshop, and more likely to provide useful feedback after the workshop. Beta pilot instructors may be lesson reviewers, Maintainers, Curriculum Advisors, or any Carpentries community member other than lesson authors. In fact, recruiting beta pilot instructors who are already playing active roles in the lesson is likely to be fruitful, as these people are invested in bringing the lesson to maturity. For two beta pilot workshops, you will need at least four instructors. Lesson authors should plan to meet virtually with pilot instructors before the workshop to answer questions and provide any technical help with setup.

6.6 Instructors

Carpentries lessons are designed to be teachable by any certified Instructor with the appropriate domain experience. In many cases, there will already be Instructors with the necessary background in our community. To help these Instructors prepare to teach the new materials, we recommend preparing an onboarding video to introduce the structure of the lesson, the dataset used, and any technical specifics that make this lesson different from other lessons Instructors are likely to be familiar with. Onboarding videos should be made available on The Carpentries YouTube channel with accurate (not auto-generated) subtitles. For inspiration, check out the existing onboarding videos in our Curriculum Onboarding playlist. To raise awareness of your curriculum, you can also schedule themed Community Discussions around your lesson.

If your lesson reaches a new domain which isn't well represented in The Carpentries community, and you would like to build capacity for teaching your lessons within The Carpentries, get in touch with team@carpentries.org.

6.7 Summary

Role	Number needed	When needed
Authors	1+	from the beginning
Reviewers	2-4 per 4 hours of content (including co-authors)	before beta pilots
Maintainers	3+ per lesson	before beta pilots
Curriculum Advisors	5-8	from the beginning (ideally) but at least before lesson publication
Beta Pilot Instructors	4+	for beta pilots
Instructors	the more the merrier!	after beta pilots

Chapter 7

The lesson life-cycle

7.1 Scope of this chapter

This chapter focuses on lessons that are being developed with the intention of becoming officially supported Carpentries lessons within an existing Lesson Program. Lessons being developed for unofficial community use may go through some or all of the stages discussed here, at the discretion of the lesson's authors. Materials intended to become part of a **new** Lesson Program must meet additional requirements as described in the Lesson Programs section of The Carpentries Handbook.

7.2 Overview and definitions

Before being adopted as an official Carpentries lesson, new lessons go through a series of stages designed to ensure they are sufficiently documented to be teachable by instructors outside of the initial author group.

In brief: the **pre-alpha** and **alpha** stages focus on developing the lesson content, while the **beta** stage focuses on documenting the lesson, so that it can be taught by anyone with the appropriate subject knowledge.

Lessons start in the **pre-alpha** stage - this stage encompasses everything from the initial lesson idea through the first time the lesson is taught. This first draft is usually written by an individual or a small group of people. From this first draft, the original authors organize an **alpha pilot** workshop at their home organisation, and collect and incorporate feedback from learners and co-instructors. They go through this iterative process a few times to bring the lesson to where it is ready to be taught by other members of The Carpentries community.

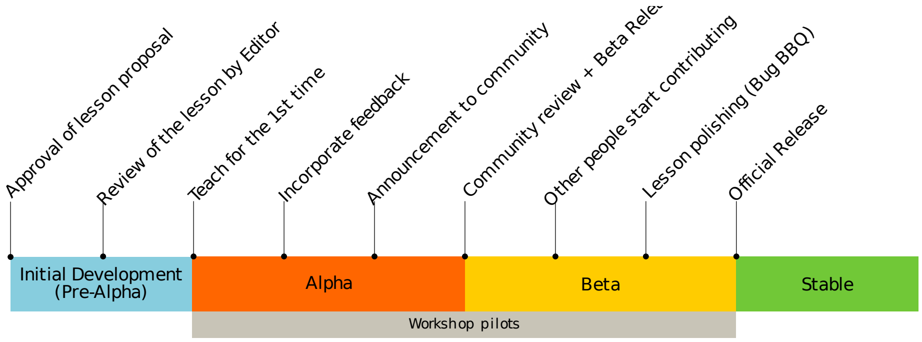

Approval of lesson proposal
Review of the lesson by Editor
Teach for the 1st time
Incorporate feedback
Announcement to community
Community review + Beta Rele
Other people start contributing
Lesson polishing (Bug BBQ)
Official Release
Initial Development (Pre-Alpha)
Alpha
Beta
Stable
Workshop pilots

Figure 7.1: Four stages of lesson release timeline. Pre-alpha stage includes approval of lesson proposal, review of lesson by Editor, and teaching for the first time. Alpha stage inclues incorporating feedback, announcement to the community, community review, and beta release. Beta stage includes other people start contributing and lesson polishing (Bug BBQ). After official release, lesson advances to stable stage.

When it is ready for broader teaching and contributions, the lesson is published for the first time on Zenodo and is now in the **beta** stage. The Carpentries staff will assist the lesson authors in organising **beta pilots**. **Beta pilots** should be hosted at a different organisation, and ideally in a different country than the alpha pilots. There are generally two or three beta pilots over a period of six months.

After beta pilots, the lesson authors and Maintainers incorporate feedback and produce a polished version of the lesson. It is now mature enough and documented enough so that anyone interested can teach it. The lesson is published on Zenodo and listed on the Lessons page for the appropriate Lesson Program. It is added to The Carpentries workshop request form, and becomes an officially supported lesson. The lesson is now considered **stable** and will remain in this stage for as long as it has active support from its Maintainer team.

If you require access to cloud compute resources to provide environments for instructors and/or learners to use in pilot workshops, for example if the lesson content involves specialised compute infrastructure or a very large data set, The Carpentries may be able to provide some support. Contact team@carpentries.org to discuss the options in such cases.

Lessons with grant support may be eligible for support from The Carpentries staff in some or all of these stages. If you are pursuing grant funding for lesson development, please get in touch with us at team@carpentries.org to discuss opportunities for staff support.

7.3 Where to start

Before you start developing a new lesson, check to see if there are already people working on creating a lesson for this topic. The Carpentries Incubator is where our community develops new curricula together. New lessons are suggested for development and added to the Incubator via the Proposals repository, so this is the best place to discuss lesson ideas and find collaborators. You can check existing issues on that repository or start a new issue if you don't see any discussions on your lesson topic. The issue template has a short set of questions for you to answer. Your answers to these questions will help us to determine an appropriate next step for your lesson materials or idea. It's a good idea to also post to our discussion forum and general Slack channel to point interested people to your Incubator issue.

Once you've submitted an issue to The Incubator, you will be directed towards one of the following tracks:

1. the Official Track
2. the Community Track
3. the Carpentries Lab Track

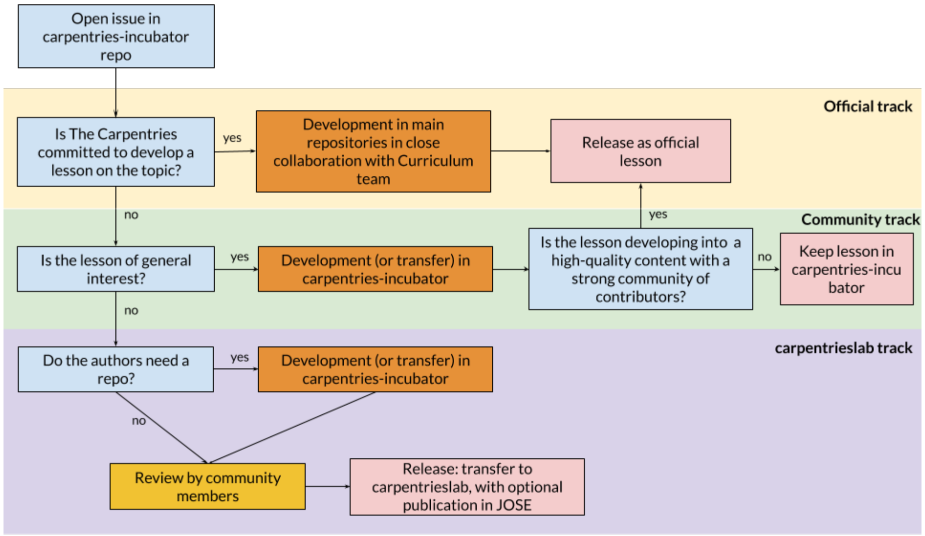

Open issue in carpentries-incubator repo
Official track
Is The Carpentries committed to develop a lesson on the topic?
yes
Development in main repositories in close collaboration with Curriculum team
Release as official lesson
no
yes
Community track
Is the lesson of general interest?
yes
Development (or transfer) in carpentries-incubator
Is the lesson developing into a high-quality content with a strong community of contributors?
no
Keep lesson in carpentries-incu bator
no
carpentrieslab track
Do the authors need a repo?
yes
Development (or transfer) in carpentries-incubator
no
Review by community members
Release: transfer to carpentrieslab, with optional publication in JOSE

Figure 7.2: Flowchart of three lesson development tracks. All lessons start with opening issue in carpentries-incubator repository.

7.3.1 Official Track

If The Carpentries have committed to develop a lesson on this topic (usually through grant funding), our Curriculum Team will work closely with you from pre-alpha through stable release, providing support on each step of the process. Your lesson will be developed in one of the official Lesson Program GitHub organisations and released as an official lesson.

7.3.2 Community Track

If The Carpentries has not committed to develop a lesson on this topic, but the lesson is potentially of general interest to our community, the lesson authors will complete the development process independently. You will develop your lesson in the Carpentries Incubator and it will be made available through the Community Developed Lessons page. If the lesson attracts a strong community of contributors, it will be considered for adoption as an official Carpentries lesson.

7.3.3 Carpentries Lab Track

This track is available for lessons on the Community Track that do not attract a strong contributor community. Lesson authors will be able to submit their materials for peer-review. After the peer-review process, the lessons will be hosted in The Carpentries Lab and will be officially endorsed by The Carpentries as high-quality resources.

7.4 Early development (pre-alpha through alpha)

We will create a repository for you in the appropriate GitHub organization, using The Carpentries lesson template. You will use the guidelines in the first five chapters of this handbook to develop your content. The Curriculum Team will be available to answer questions about the template and provide pedagogical guidance. For lessons on the Official Track, authors will meet regularly with a member of the Curriculum Team to work through the lesson drafting process.

Lessons on the Community Track will be developed in the Carpentries Incubator. This Appendix aims to provide information and further guidance for Incubator lesson developers.

7.5 Field testing: alpha stage

Once your lesson is ready to be taught for the first time, it will enter the **alpha stage**. Field-testing a lesson is a good opportunity to receive and incorporate feedback from learners, instructors, and workshop helpers who can compare their expectations to reality. The initial feedback gathered during these first workshops is really important.

During alpha pilot workshops, instructors and helpers should take detailed notes, including:

- amount of time used to teach each section
- amount of time used for each exercise
- technical issues that arose during installation
- bugs or parts of the lesson code that didn't work as expected
- incorrect or missing exercise solutions
- questions learners asked (and their answers)
- parts of the lesson that were confusing for learners

These notes can be collected in an Etherpad, Google Doc, or other collaborative document that is shared with co-instructors and workshop helpers. This document should not be shared with workshop learners, as it would add significantly to their cognitive load.

After the workshop, instructors should share the notes document with the lesson authors - who will convert the notes into individual issues in the lesson repo. For two-day workshops, lesson authors should expect to spend at least four hours to create follow-up issues, and at least eight hours to putting in PRs to fix these issues. For two pilot workshops, this translates to ~24 hours of work, which can be distributed among the lesson authors.

After incorporating workshop feedback, the lesson is now ready to be published. For lessons on the Official Track, our Curriculum Team will create a publication record on Zenodo. Lesson authors on the Community or CarpentriesLab tracks may also choose to publish their lesson on Zenodo. At this stage, the lesson is in **beta**.

The Carpentries Community Handbook contains guidance and templates to help you organise and run pilot workshops.

7.6 Polishing: beta stage

Now that the lesson has been published, it is ready for teaching by instructors outside of the original authorship team, and for contributions from the broader Carpentries community. For lessons on the Official Track, Carpentries staff will assist lesson authors in recruiting instructors to teach beta pilots. Authors for non-Official Track lessons can recruit beta pilot hosts and instructors through our discussion list, Slack organization, on Twitter, and through domain-specific mailing lists or other community groups they are part of. For more information about the role played by and qualifications needed to be a beta pilot instructors, see our chapter on community development roles.

The **beta stage** lasts approximately 6 months. During this time, members of The Carpentries community can teach it and contribute to the content of the lesson. The main goal of this phase of the lesson development is to develop the documentation needed to ensure that people who have not contributed to the initial development efforts of the lesson have enough information to teach it effectively.

Beta pilot instructors and helpers should take notes similar to those described above for alpha pilots, but should (hopefully!) notice fewer bugs and other issues. As with alpha pilots, beta pilot instructors will provide the lesson authors with the notes they've collected during the workshop and the authors will convert those notes to issues and PRs to resolve concerns raised during the workshop. Maintainers and other community members will be involved with this clean-up phase as well.

7.7 Stable stage

After the polishing process, the lesson is ready to be marked as stable.

7.7.1 Official Track

Lessons that have completed beta testing go through a final check by a member of the Curriculum Team, before being published again on Zenodo, this time in its **stable state**.

Official Track lessons will be added to the appropriate Lesson Program lessons page (e.g. Data Carpentry's lessons page) and to our workshop request form. Anyone may now request a centrally-organized Carpentries workshop using this curriculum. Stable official lessons are also eligible for trainee Instructors to teach from in a teaching demonstration for their Instructor Training checkout.

The Curriculum Team will recommend suitable starting points for teaching demonstrations using the lesson, and will submit these to the Instructor Training Leadership for and review and addition to the list of approved lessons for demonstrations.

7.7.2 Community/Carpentries Lab track

For lessons on the CarpentriesLab track, authors will submit their stable stage lessons for review. The CarpentriesLab Editor will select 2-3 reviewers within The Carpentries community with teaching experience and/or appropriate domain expertise, who will provide an open and friendly review of the lesson. After incorporating feedback and comments from the reviewers, your lesson will be badged "Reviewed by the Carpentries Community" and will be listed on our websites as such. During this process, you will have the possibility to include a short paper describing your lesson in the GitHub repository and have your lesson considered for publication in JOSE, the Journal of Open Science Education. **This review process is in development and we are not yet accepting submissions. Please watch for announcements as we roll out this program.**

7.8 The stable lesson: Maintenance and lesson releases

Congratulations! Your lesson has now been published and is being actively taught by the community. That means you're done, right? Not exactly. In order to ensure that your wonderful lesson materials continue to be relevant and useful, you'll need to build and train a team of lesson Maintainers, who are responsible for day-to-day upkeep and periodic publication of the lesson.

In a previous chapter, we discussed the process for recruiting and training Maintainers. Here, we will focus on the responsibilities of Maintainers, their day-to-day workflow, and their involvement in the lesson release process.

As your lesson is taught, workshop instructors and helpers will post issues and pull requests to the lesson's GitHub repository. These will range from typo corrections, to installation problems, to recommendations for changing the libraries or function calls demonstrated in the lesson, and will therefore require different levels of consideration and technical expertise to implement. A typo correction will probably be taken care of independently by a single Maintainer, while a proposal to change the plotting system used in the lesson from matplotlib to seaborn will require significant discussion. Each lesson Maintainer team will develop their own strategies for managing their work, but we recommend the following as a starting point.

- Maintainers for a lesson should set a regular meeting to discuss any unresolved issues that have arisen and to decide on the division of responsibilities until their next meeting.
- One Maintainer commits to monitor the repo daily and respond to new issues and PRs to acknowledge them, thank the contributor, and apply appropriate issue tags.

- Once a week, one Maintainer looks at the list of active issues and PRs and assigns each to one team member, based on the time commitment and availability discussed at that month's meeting.
- Individual team members work on resolving issues and PRs, as assigned, asking their co-Maintainers for review when needed.

The Maintainer community is working on developing a set of guidelines and template language for responding to common situations. These guidelines will be added to this section when they become available.

The above description focuses on Maintainers' ongoing duties. Maintainers are also involved with lesson releases, which take place roughly every six months. Information about lesson releases will be added to a future version of this handbook.

Chapter 8

Technological introductions

Our goal is to make developing and contributing to lessons as simple and accessible as possible. The more people who are able to contribute to a lesson, the more we can harness community knowledge and experience to create materials that are teachable and applicable in a range of learning contexts. We recognize that technology can be a major barrier to contribution, and we are currently working on a major update to our lesson infrastructure to reduce this barrier. This chapter reflects our current lesson infrastructure and describes what lesson authors and other contributors need to know to use The Carpentries lesson template. In the individual sections below, we have noted whether particular technological knowledge will remain necessary or be phased out in upcoming infrastructure updates.

What knowledge are we assuming lesson authors will have? What knowledge will this chapter help them gain?

8.1 Lesson hosting and rendering

The Carpentries hosts all of our lessons on GitHub. We use a shared lesson template to provide aesthetic and structural consistency across our lessons. Template files for each lesson are rendered into a webpage using Jekyll - a static site generator which is written in Ruby.

You do *not* need to know Ruby or Jekyll to write or contribute to a Carpentries lesson. However, you will need to have these software packages installed on your computer if you want to view your lesson materials locally before pushing them to GitHub.

Information about installing Ruby and Jekyll is available in the APPENDIX.

8.2 Using the lesson template

Each lesson is made up of *episodes*, which are focused on a particular topic and include time for both teaching and exercises. A lesson repository (or "repo") includes one file for each episode, and a set of helper files that are required to build the lesson webpage. Most of these helper files are standardized across all of our lessons and aren't something that lesson authors or contributors need to interact with. In this section, we will focus only on the files that you are likely to interact with. If you're interested in the details of how the template is structured, and what each of the files does, these details are provided in APPENDIX. We recommend not spending time learning these details now, as we are in the process of greatly simplifying our lesson template.

8.2.1 Creating your lesson repository

The following sections will guide you through the pieces of the lesson template that you will need to modify to create your lesson. In order to follow along with these examples, you can start by creating your lesson repository in GitHub. To do this, follow the setup instructions on our example lesson.

8.2.2 Lesson homepage

The lesson homepage provides an overview of the lesson, including any prequisites, an introduction to the dataset used, a schedule showing the episodes and the time alloted for each, and any other information learners will need for the lesson.

For inspiration to guide you in writing your lesson homepage, check out these examples for Data Carpentry, Software Carpentry, and Library Carpentry.

The lesson homepage is built from the `index.md` file, which is automatically created when you initialize a lesson repository. You will need to add the following to this file:

1. A few paragraphs of explanatory text describing the lesson.
2. One or more `.prereq` boxes detailing the lesson's prerequisites, giving an overview of the dataset, and/or calling attention to the lesson's Instructor notes.

The schedule will automatically be included in the lesson homepage based on information present in the episode files.

8.2.3 Episode files

The majority of a lessons content is in its episode files. Episode files are stored in the `_episodes/` folder within your lesson repo (or in `_episodes_rmd/` for lessons written in R). Episode file names must start with a two-digit identifier number (e.g. `01`) followed by a short descriptive name, separated by a dash (`-`). For example `02-loop.md`, `03-lists.md`. The numeric identifier is used to place your episode files in the correct sequence within the lesson. Episode files are written in *Markdown* (more on that in a moment) or *RMarkdown*.

8.2.3.1 Episode headers

When your lesson repository is created, it will start out with one pre-made episode file (`01-introduction.md`). You can use this file as a template for creating each of your episode files, as it provides an example of how these files must be structured. The content of this pre-made episode file is shown below:

```
---
title: "Introduction"
teaching: 0
exercises: 0
questions:
- "Key question (FIXME)"
objectives:
- "First learning objective. (FIXME)"
keypoints:
- "First key point. Brief Answer to questions. (FIXME)"
---
FIXME

{% include links.md %}
```

The material between the first and second instances of `---` is called the *YAML header*. The information stored in the YAML header is used by the lesson template to populate important parts of the lesson webpage. This section explains each component of the YAML header and what that information is used for.

For each episode, you will need to create a copy of this file and:

1. Replace `Introduction` with the episode title (**not** the lesson title) in quotation marks. The episode title will appear on the episode page and in the schedule that appears on the lesson homepage.
2. Enter an estimated number of minutes for teaching the episode and an estimated number of minutes for learners to spend completing challenge problems (including class discussion of challenge solutions). These time

estimates will likely be updated by Instructors as they get real-world data on how learners respond to the pacing of the episodes, but it is useful to have a starting point to benchmark from. The lesson template creates a schedule from these time estimates and places it on the lesson homepage.

3. Replace `Key question (FIXME)` with 1-3 motivating questions for the episode, each on a new line and in quotation marks. These motivating questions will appear in the schedule on the lesson homepage.
4. Replace `First learning objective. (FIXME)` with 3-7 learning objectives for the episode, each on a new line and in quotation marks. For information on writing useful learning objectives, see the Developing Content chapter.
5. Replace `First key point. Brief Answer to questions. (FIXME)` with 3-7 major take-aways from the episode. For information on how to distill an episode's key points, see the Developing Content chapter. Key points for all episodes are shown together in the lesson's reference page.

8.2.3.2 Episode content

After the YAML header, your episode file will contain the content for that episode. This content will likely include:

- paragraphs of text
- lists
- tables
- images or figures
- code chunks
- special blockquotes, including exercises and solutions (described below)

This content will be written in *Markdown*, a light-weight markup language that makes it possible to create fancy HTML pages using only a few formating tricks. In this section, we'll cover only the Markdown syntax that you will need in order to create the content types listed above. You can find more information about Markdown at https://commonmark.org/help/.

1. Paragraphs of text - To create text paragraphs in Markdown, just type as you normally would! A few neat tricks:

- surround text with a single pair of stars (`*`) to make text *italic* (`*italic*`)
- use a double pair of stars to make text **bold** (`**bold**`)
- create headers by starting a line of text with two hash signs (`##`) There are lots of other fancy things you can do, but this should get you started!

1. Lists - To create a numbered list in Markdown, do this:

```
1. A
1. numbered
1. list
```

This will show up like this:

1. A
2. numbered
3. list

Hint: You can use sequential numbers if you want, but it's easier to update the list later if you use only 1s. Markdown will create the sequence for you.

To create an un-numbered list in Markdown, do this:

```
* An
* unnumbered
* list
```

This will show up like this:

- An
- unnumbered
- list

1. Tables - To insert a small table into your episode, do this:

```
| Category | Item |
|--------- | ---- |
| Food     | Sandwich |
| Drink    | Tea |
| Food     | Apple |
```

This will show up like this:

Category	Item
Food	Sandwich
Drink	Tea
Food	Apple

1. Images or figures - Place a copy of the image you would like to display into the `fig` directory. You can then link to the figure using the syntax:

```
![Figure Description](../fig/figure_file_name.svg)
```

Note about alt text. Question: Is this the "proper" way to insert a figure link?

1. Code chunks - As discussed in an earlier chapter, Carpentries workshops are taught using participatory live coding. Instructors type the code as they teach it and learners type along with the Instructor. For more information about how live coding works, and what its advantages and disadvantages are, read that section of our Instructor Training program. The fact that Carpentries workshops are taught using live coding means that much of your episode content will be *code chunks* - short blocks of code that learners type along with the Instructor and evaluate on their own machines. Each code chunk should correspond to one interactive session. If learners will be running the code as two distinct commands, that code should be displayed as two distinct chunks in the episode file.

You can add a code chunk to your episode using the following syntax:

```
~~~
pwd
~~~
{: .source}
```

Which will show up like:

```
pwd
```

Code chunks that learners should type out with the Instructor should use the `{: .source}` tag as shown above. Chunks that show expected output should use the `{: .output}` tag. Chunks that show an expected error message should use the `{: .error}` tag.

The generic `{: .source}` tag can be used for all programming languages. To make your code more stylish, you can use a language-specific tag (instead of `{: .source}`). This will add things like syntax highlighting to your code. The language-specific tags available with our lesson template are:

- `{: .language-bash}`
- `{: .html}`
- `{: .language-make}`
- `{: .language-matlab}`
- `{: .language-python}`
- `{: .language-r}`
- `{: .language-sql}`

You don't need to use these language-specific tags, but they make your lesson a little prettier.

1. Special blockquotes - Exercises, solutions, helpful tips, and a few other types of special information are formatted as blockquotes within the episode file. Each blockquote has the same general structure, but ends with a different tag. The ending tag determines how the blockquote will appear on the lesson webpage. The general structure of a blockquote is:

```
> ## Title
>
> text
> text
> text
{: .callout}
```

where the `{: .callout}` tag should be replaced with one of the following as appropriate:

- `{: .callout}` for sharing an aside or comment. Use sparingly.
- `{: .challenge}` for an exercise.
- `{: .discussion}` for a discussion question.
- `{: .solution}` for an exercise solution.

Additional blockquote tags included in our lesson template are described in APPENDIX, however, the four listed above should cover all normal use cases for a lesson author.

Exercise solutions are nested within the blockquote for that exercise, as shown below:

```
> ## Challenge Title
>
> This is the body of the challenge.
>
> > ## Solution
> >
> > This is the body of the solution.
> {: .solution}
{: .challenge}
```

Code chunks may also be nested within blockquotes as needed.

8.2.4 Extras

So far we've covered how to create and format the content of your lesson homepage (in the `index.md` file) and your lesson episodes (in multiple `.md` files within the `_episodes` directory). This covers most of the files you will need to work with when you create a new lesson. There are a few remaining files that you will need to populate in order for your lesson to be fully fleshed out and ready to teach:

1. The Reference page (`reference.md`) - this file will be created automatically and will include a list of all of the keypoints that you defined in your episode YAML headers. You don't need to do anything to create this list! However, it's a good idea to add a glossary of terms that are used in your lesson.

2. The Setup page (`setup.md`) - this file will be created automatically, but needs to be populated with installation instructions for software learners will need to have before begining the lesson. If learners are expected to download data prior to the workshop, that data should also be linked and described here. The setup page may be quite simple or more complex, but should always include installation information for all three major platforms (Windows, Unix/MacOS, Linux).

3. The Instructors' Guide (`_extras/guide.md`) - this file should provide additional discussion useful to instructors, but not appropriate for inclusion in the main lessons. Remember not to overload on details, and to keep the information here positive and useful for instructors! This guide should include the following sections:

- Lesson motivation and learning objectives - These concepts should be highlighted in the main lesson material, but ideas for explaining these concepts further can be placed here.

- Lesson design - Most lessons contain more material than can be taught in a single workshop. Describe a general narrative (with time estimates) for teaching either a half day or full day with this lesson material. You may also choose to include multiple options for lesson design, or what material can be skipped while teaching. This section may also include recommendations for how this lesson fits into the overall workshop.

- Technical tips and tricks - Provide information on setting up your environment for learners to view your live coding (increasing text size, changing text color, etc), as well as general recommendations for working with coding tools to best suit the learning environment.

- Common problems - This can include answers to common learner questions, as well as links to resources (blog posts, stack overflow answers, etc) that may solve problems that may occur during a workshop.

8.2.5 Other information

In addition to populating the lesson homepage, lesson content (episodes), Instructors' guide, and other lesson-specific pages, lesson authors need to make a few changes to the template files to make sure that the lesson has all of the neccessary information.

1. In the `_config.yml` file,

- set the `carpentry` variable to the appropriate lesson program,
- set `title` to be the overall title for your lesson,
- set `email` to the correct contact email for your lesson.

1. In the `CONTRIBUTING.md` file, change the `issues` and `repo` links to match the URLs of your lesson.
2. In the `CITATION` file, add information about how to cite your lesson.
3. The `AUTHORS` file should include a list of the lesson's authors.

8.2.6 Special notes on RMarkdown

It is also possible to write episodes in RMarkdown, to be executed and rendered as pages via an automated step in the lesson template configuration.

To learn more about how to incorporate RMarkdown documents as episodes in a lesson, please read the Using RMarkdown section of The Carpentries Lesson Example. The Lesson Example page assumes the reader is already familiar with the Rmarkdown framework. For those wishing to get started with writing in RMarkdown, this online book provides an excellent introduction.

8.3 Working on GitHub

GitHub is a web-based service for hosting code under version control. In addition to being a technical platform, GitHub is also a social media platform and has its own standards around etiquite and interaction. This section describes how The Carpentries community tends to interact on GitHub and gives you some tips for navigating this new social scene.

All Carpentries lesson materials, whether they are established or in early development, are hosted publically on GitHub in one of several *organisations*. The following high-level organisations are managed by The Carpentries:

- carpentries - hosts The Carpentries website and materials for programs that span individual lesson programs, such as our Instructor Training curriculum and The Carpentries Handbook
- datacarpentry - hosts Data Carpentry specific lesson materials and website
- librarycarpentry - hosts Library Carpentry specific lesson materials and website
- swcarpentry - hosts Software Carpentry specific lesson materials and website
- data-lessons - hosts lessons in development which are targeted to become part of the official Carpentries lesson stack
- carpentrieslab - community-developed lessons which may or may not become part of the official Carpentries lesson stack

In order to contribute to lesson materials, you will need a GitHub account.

To manage changes, we follow GitHub flow. Each lesson has two maintainers who review issues and pull requests or encourage others to do so. The maintainers are community volunteers and have final say over what gets merged into the lesson. To use the web interface for contributing to a lesson:

1. Fork the originating repository to your GitHub account.
2. Within your version of the forked repository, move to the default branch (e.g. `gh-pages`) and create a new branch for each significant change being made.
3. Navigate to the file(s) you wish to change within the new branches and make revisions as required.
4. Commit all changed files within the appropriate branches.
5. Create individual pull requests from each of your changed branches to the `gh-pages` branch within the originating repository.
6. If you receive feedback, make changes using your issue-specific branches of the forked repository and the pull requests will update automatically.
7. Repeat as needed until all feedback has been addressed.

When starting work, please make sure your clone of the originating `gh-pages` branch is up-to-date before creating your own revision-specific branch(es) from there. Additionally, please only work from your newly-created branch(es) and *not* your clone of the originating `gh-pages` branch. Lastly, published copies of all the lessons are available in the `gh-pages` branch of the originating repository for reference while revising.

If you choose to contribute via GitHub, you may want to look at How to Contribute to an Open Source Project on GitHub.

Appendix A

The Carpentries Incubator

The Carpentries Incubator is where The Carpentries community develops curricula together. It is intended to serve as a place to find and contribute to lessons that members of our community are developing with the approaches and infrastructure introduced in this handbook.

A.1 Lesson Proposals

Proposals to begin development of a new lesson in the Incubator, and to transfer existing material (regardless of its current state/stage of development) into the Incubator, can be made by opening an issue on the Proposals GitHub repository. The requirements are minimal for a proposal to be accepted, predominantly designed to maintain consistency across the community's lessons. The README of that repository contains more detailed guidance for those who wish to submit a proposal.

A.2 Topic Tags

Lessons under community development in the Carpentries Incubator are listed on The Carpentries website, based on metadata describing the lesson. This metadata is added in the form of topic tags on the lesson repository. These topic tags should be set as soon as possible after the lesson has been created or added to the Incubator. Some are essential and taken from a limited set of values, while others are more flexible. See the table below for guidance about the types and number of topic tags each lesson repository should have.

Category	Example	Number	Description
Lesson*	`lesson`	1	Must be `lesson` to be listed on the Community Developed Lessons page.
Location*	`carpentries-incubator`	1	`carpentries-incubator` or `carpentrieslab`.
Language*	`español`	>0	The language(s) the lesson is available in.
Stage*	`alpha`	1	The current development stage for the lesson. See Chapter 7 for more info.
Domain	`microbial-ecology`	1-2	The high-level domain(s) of the lesson, for a general categorization.
Tools	`python`	1-3	The main tool(s) taught in the lesson.
Skills	`taxonomic-classification`	1-3	The main skill(s) taught in the lesson.

Categories marked with an asterisk (*) are required in order for a lesson to appear and be appropriately sorted on the Community Developed Lessons page.

The Curriculum Team will support you in setting appropriate topic tags for your lesson. To help ensure consistency across all lesson repositories developed by The Carpentires community, please refer to this listing of topic tags currently in use in The Carpentries Incubator, and re-use these values where appropriate, creating new topic tags where no pre-existing label exists for your lesson.

A.3 Using GitHub Issue Labels

GitHub allows the maintainers of a repository to add contextual information to Issues and Pull Requests in the form of labels. Visit the How to Label Issues page of The Carpentries handbook for a list of the labels used on official Carpentries lesson repositories and to read recommendations on how to use them for your own lesson.

A.3.1 Using Labels to Encourage Contributions

Two labels, used by The Carpentries and in many repositories across GitHub, can be deployed to increase the visibility of your lesson and encourage community members to contribute to its development.

The **“help wanted”** label should be used to highlight issues with which you would welcome additional help. The Carpentries website includes a Help Wanted page, which can automatically list every issue labelled “help wanted” on repositories from The Carpentries, Software Carpentry, Data Carpentry, Library Carpentry, CarpentriesLab, and The Carpentries Incubator. Find out how to include issues from your lesson repository on the Help Wanted page by reading the Information for Maintainers on the page itself.

The **“good first issue”** label should be used to identify issues that would make a good entry point for newcomers looking for a way to contribute to your lesson. The work needed to close an issue with this label would typically not require an extensive knowledge of the structure or intricacies of your lesson repository, or an expert understanding of the content. The “good first issue” label is used so extensively that GitHub provides a page at `[repository URL]/contribute` for every repository, listing issues with this label.

Appendix B

Bibliography

Appendix C

Adapting Existing Lessons for The Carpentries

A guide for those wishing to adapt existing lesson materials, written/published in formats other than The Carpentries lesson template, to conform to the requirements for inclusion in a Carpentries lesson program or The Carpentries Incubator.

C.1 Lesson Template

Below are some suggested approaches to incorporate previously-written lesson content into The Carpentries lesson template. Unfortunately we cannot provide detailed instructions or automated conversion for content in every format, but this section includes suggested workflows to adapt content in several of the most common formats for lessons teaching software and data skills.

C.2 Conversion from Jupyter

Jupyter notebooks in `.ipynb` format can be converted to Markdown via *File* -> *Download as* -> *Markdown (.md)* in the graphical Notebooks interface or by using the `nbconvert` command line tool.

The `convert_ipynb.py` Python script, based on the `convert.py` script written by Allen Downey while developing the Data Carpentry Astronomy curriculum, processes Jupyter Notebooks such that, afterwards, the Markdown derived from them can be more readily incorporated into The Carpentries Lesson Template. The script makes some assumptions about what your Jupyter notebooks will look like (e.g. it looks for third-level headings beginning “Exercise” to define exercise blocks), but it may save you some time if you have a large number of notebooks containing a lot of code blocks etc.

To be rendered as episodes of a lesson site, the resulting Markdown generated from the `.ipynb` files processed by `convert_ipynb.py` will need to have YAML front matter added, before being placed in the `_episodes` directory of a lesson repository (see Markdown below).

C.3 Conversion from RMarkdown

Lesson content written with RMarkdown can be directly included in the lesson template, by adding RMarkdown episodes to the `_episodes_rmd` directory.

The *Using Rmarkdown* episode of the Lesson Example provides more details of how to write episodes in RMarkdown. Briefly, the important points are:

1. You must add the episode front matter just as you would for standard Markdown.
2. You should commit your RMarkdown files to the `main` branch. The lesson build workflows have been configured such that any RMarkdown files commited to the `_episodes_rmd` directory in the `main` branch of the GitHub lesson repository will be converted to standard Markdown and copied to the `_episodes` directory of the `gh-pages` branch, where the content will be served by GitHub Pages.
3. You should use the `fig.alt` parameter to define the alternative text property for any figures/images created by code chunks in your RMarkdown.
4. You will need to write the lesson landing page (`index.md`), installation/setup instructions (`setup.md`), Instructor Notes (`guide.md`), and glossary/reference guide (`reference.md`) in standard Markdown.

C.4 Conversion from other formats

Pandoc is a universal format conversion tool, which may be able to help when converting other formats (e.g. ReStructured Text) to Markdown.

C.5 Markdown

Whether you have lesson content already in Markdown (e.g. from GitBook) or converted to Markdown from a Jupyter Notebook or any other format, there are some additional steps you will need to take to make this content compatible with the lesson template:

C.5.1 Set up a lesson repository

For lessons being submitted to The Carpentries Incubator: create a new issue to the Incubator Proposals repository to request a new lesson repository. After accepting an invitation, sent by email, to join the `carpentries-incubator` organisation, you will receive full administrative access to a new lesson repository with some brief instructions to help you get started.

For lessons being developed outside The Carpentries Incubator: follow the setup instructions in the Lesson Example.

A note about the license and code of conduct: Carpentries lessons use the Creative Commons Attribution version 4.0 (CC-BY) license for lesson material, and the MIT license for any software and example code included in the lesson repository. Laws vary from country to country but, as a general rule, **changing the license of a project requires explicit agreement from all previous contributors to the repository**. You should make sure that you have obtained this agreement before you transfer the material into the lesson repository (which already includes the license file), or clearly state that you are re-using material originally created elsewhere (if the license of the existing material allows this). Similarly, **we require that all lessons follow our Code of Conduct**, and you should make sure that you and all of your collaborators have read and understand this document, and that your existing lesson material complies with the policies described in it, before you begin working in the new repository.

C.5.2 Add YAML Front Matter

The Jekyll engine that renders sites on GitHub Pages identifies Markdown files to build into webpages based on the presence of a header section, known as *front matter*, which is located at the top of the file. The *Technological Introductions* chapter details the fields that must be present in this YAML header. Note: episode files written in RMarkdown must include an additional line, `source: Rmd`, to be rendered correctly.

If your lesson already includes statements of learning objectives/outcomes, questions, and/or key points for each section, this information should be transferred to the YAML header for the respective episodes.

C.5.3 Add *alt* Text to Images

To increase the accessibility of Carpentries lessons, all images should include a short, descriptive *alternative text* property. In Markdown, alternative text (alt text) is added between the `[]` of the image definition, i.e.

```
![Image alt text](path/to/image/file.svg)
```

Accessibility consultants Deque provide a good overview of the important things to consider when writing alt text, and a Medium post from Amy Cesal gives guidance on alt text for data visualisations.

C.5.4 Add to `_episodes`

To be built by Jekyll and included in the navigation of the resulting lesson site, Markdown files with front matter should be located in the `_episodes` directory of your lesson repository. Other files needed for a lesson, but not located in the `_episodes` directory, include the lesson landing page (`index.md`), the installation/setup instructions (`setup.md`), Instructor Notes as an aid to those teaching the lesson (`guide.md`), and the glossary/reference guide (`reference.md`). You can find more complete information about these files, and others included in the lesson template, in the *Lesson Organization* episode of the Lesson Example.

C.5.5 Organise supporting files

Images used as figures in the lesson should be stored in the `fig` directory, example code in the `code` directory, and example data in `data`. Any other supporting files that do not fit into the previous three categories can be placed in `files`. Remember to adjust any internal links in your lesson to match the new locations of these files.

C.5.6 Implement formatted blocks

Some of the key visual elements of a Carpentries lesson are the boxes used for callouts, lesson prerequisites, exercises, solutions, etc. You should add the appropriate formatting to any relevant parts of your lesson, so that they render correctly. Generally speaking, this involves converting these elements to blockquotes followed by a tag that applies a CSS class to the preceding block.

See the *Formatting* episode of the Lesson Example for details and examples. That page also includes more information about the best way to display code blocks in the lesson template.

C.5.7 (Recommended) Centralise Link References

To reduce the effort associated with modifying the target URLs of links in your lesson if/when they change, you can collect all these target URLs as Markdown link references, in the format `[link-ref]: https://carpentries.org`, in the `_includes/links.md` file of the lesson repository.

You can then refer to these with the syntax `[link text][link-ref]` after adding `{% include links.md %}` to the very bottom of the page content.

C.5.8 Complete Lesson Repository Setup

To finish setting up your lesson repository

- fill in any fields marked "FIXME" in the lesson `README.md` and/or `_config.yml`
- list the names of current and past maintainers in the lesson `README.md`
- update the `AUTHORS` and `CITATION` files in the lesson repository
- add the lesson URL and topic tags to the *Description* of your lesson repository on GitHub

Appendix D

Lesson Sprint Recommendations

One way of making progress and encouraging collaboration on the development of a lesson is to organise a dedicated event such as a co-working session or a lesson development sprint. A co-working session is typically a shorter, less formal event bringing people together to work simultaneously on the lesson. The co-working session might include brief check-ins for the group to discuss progress and next steps, but most of the time will be spent working on the lesson independently or in small groups. A sprint is typically a longer, more structured event designed to welcome and bring onboard new contributors to the lesson project.

D.1 Introduction

The following set of recommendations, written by members of The Carpentries community, is intended to guide lesson authors through the process of organising a lesson sprint. The recommendations are relevant to a lesson sprint consisting of a single session or in multiple sessions with breaks between, whether on one day or multiple consecutive days. We have tried to note where a particular recommendation is only relevant or particularly relevant to a sprint being run across multiple days/sessions. These recommendations were written with a particular focus on sprints taking place virtually, i.e. with participants meeting by video conference, but many of the recommendations (especially those in the Before and After sections) are also relevant to in-person events.

D.1.1 Target Audience

The target reader is one of the lead developers/maintainers of a lesson under active development e.g. in The Carpentries Incubator. They should already have a clear idea of the target audience and main learning objectives for their lesson, even if they haven't started writing the actual material yet - see the *Planning your lesson* section for more details of the steps we recommend you take with your lesson before it is time for a sprint. They may be planning to teach the lesson for the first time, or have recently taught it for the first time, and now want to improve the lesson material based on feedback collected when teaching.

If you in the stage where you are not yet clear on the target audience or the overall learning objectives for your lesson, we suggest you work through the *Deciding what to teach* chapter of this handbook before returning to this guide.

D.1.2 Objectives

Provide practical tips and more theoretical guidance about how to:

- organise an inclusive and accessible sprint
- make the most of a dedicated lesson sprint
- create coherent lesson material from individual contributions made during the sprint
- engage contributors before, during, and after the lesson sprint

D.2 Before the Sprint

- **Establish an organising committee.** Organising the sprint will be much easier if there are multiple people involved in planning and running the event. Aim to establish an organising committee with members from diverse backgrounds who can bring a broad perspective to create an accessible and inclusive event for your participants. If you have more than 2 people, be sure to assign roles and responsibilities to avoid diffusion of responsibility.

- **Select a code of conduct.** If your lesson repository already has a code of conduct, apply it to the sprint and associated events too. If your lesson repository does not already have a code of conduct, add one. Like licenses, we do not recommend that you write your own code of conduct from scratch. Instead, re-use/adapt an existing text you find online. We recommend The Carpentries Code of Conduct, which was designed with exactly these kinds of project in mind. Assign at least one person as the contact person/facilitator for matters relating to the Code of Conduct, and make sure all sprint participants are aware of who this is and how they should connect with them. You can do this by including their details in the "primer" information you provide to participants before the event and by introducing them when you mention the code of conduct at the start of the sprint itself.

- **Set up a planning meeting/call for the committee** Use this meeting to agree on the dates, event format and the target audience in sufficient detail and high-level objectives of your sprint. Document your discussions and plans in a central location so that everyone involved in the process can access them. Divide responsibilities to put together resources (such as for communication and norms) that will be shared with all participants. (See below for more details.)

- **Consider participants' roles and expertise.**

  – Do you have a team with diverse skillsets and topic knowledge required to meet your high-level objectives? If not, consider finding and inviting other people.

- **Establish communication channels and norms.** This will set an expectation for how collaborators will communicate before, during, and after the sprint.

  – Dedicated messaging channels have been used with success for conversations/short updates during the sprint and in-between sprint conversations. See During the Sprint for more details.
  – Mailing lists for initial organisation and other important and more permanent communication, e.g. scheduling meetings.
  – [**Online**] Video conferencing technology that is an accessible choice for everyone and suitable for the format of the sprint.
  – Decide on the basic etiquette for participation, such as how to collect ideas, who to contact for support, how to participate in an ongoing discussion, how to contribute to content development and how to review contributions.
  – Choose a collaborative note-taking platform for recording and communicating decisions, minutes and actions from meetings. (See the tools section for details of some different note taking platforms.)

- **Think about the timing of your sprint.**

  – You can run the sprint in conjunction with an upcoming event, e.g. a conference/meeting that many in your community will already be engaged with. If you would like to attract new contributors, this will make it easier to get people's attention and convince them to block time in their calendar to join in with the sprint. You may also be able to tap into the promotional communications etc. for that event,

    and/or invite people to join the sprint, e.g. via a lightning talk or poster.
  - Conversely, you may want to avoid clashes with another event if your team is going to be engaged elsewhere or have other commitments.
  - Announce the dates for your sprint at least 6 weeks in advance.

- **Add dates and time in your calendar**: In addition to the event dates, schedule a few planning meetings with the committee members *before* the sprint and one meeting/co-working session *after* the sprint for wrap-up/follow-up tasks.

  - When possible, offer a pre-event onboarding call that can be used for sharing all the relevant resources and setting expectations around what to expect at the event.

- **Scope out what content will be worked on.** Defining objectives and which sections/aspects of the lesson should be worked on during the sprint will reduce the energy/time required for contributors to begin working and help to keep efforts focused during the event. This guidance might be in the form of a list on a shared document or open issues on the lesson repository and do not need to contain much detail. You do not need to micromanage individual contributions, but a summary of each key objective can serve as a starting point for discussions between collaborators about the best approach to reach those goals.

- **Create a resource for good practices for lesson design.** This resource should be shared with all the participants to assist them in designing lessons to be as effective as possible. Consult the rest of this Handbook for recommendations that you and your collaborators can follow when creating the lesson material during the sprint.

- **Prepare a video call link for all sprint sessions.** [**Online**] If your sprint is taking place across multiple sessions, or will include multiple video calls between participants, try to provide a single meeting link for them all. This will save contributors time when returning to the sprint calls, e.g. for a check-in session or after a break.

- **Communicate participation information and send reminders before the Sprint**

  - Send short reminders a week and a day before the Sprint, with time and date, video conferencing URL, note docs and any prior preparation work
  - If prior reading/preparation is required, give at least a week lead time
  - Think of a way to let participants withdraw their participation/leave early/join late comfortably; this helps especially for folks juggling with multiple responsibilities

- **Use time zone-converted event links for remote participants.** [**Online**] For sprints running online with the potential for participants from multiple regions/time zones, it is vital to announce time(s) and date(s) for your sprint in a way that allows visitors to see the equivalent in their local time. Always list event timings in UTC and use tools like timeanddate.com to provide an automatic conversion to local time.

- **Promote the sprint widely.** You might want to promote your upcoming sprint to attract contributors beyond your current team.

  - You can use mailing lists, Slack channels, Newsletters, Twitter and direct invitation to reach your desired participant groups.
  - When possible, offer financial support to people who may want to participate but may have special needs to facilitate their participation for the duration of the event (for example, childcare for in-person, high-speed internet for remote events).
  - Consider whether you need a system to track participation for the Sprint, e.g. a Google Form/Eventbrite page
    - You may not need this if you already have dedicated communication channels (see "establish communication channels and norms") and the Sprint is open to everyone in those channels
    - You may want to collect additional information from participants, e.g. if there are multiple parallel discussion topics, which discussion they'd like to participate in; dietary requirements and emergency contact information for in-person events; accessibility needs and time zone for virtual events
    - Tools like Eventbrite allow organisers to easily send emails to all attendees, e.g. reminders and follow-ups.

- **Prepare a "sprint primer" document and publish it.** Provide participants in advance with a single document that provides all the information they need to know before joining (or deciding to join) the sprint: what steps do they need to take to register, to attend/connect, to prepare? What can they expect when they join? What will the format be? What roles will there be within the group? In what different ways can participants contribute during the sprint? What channels will be used for communication? What is the schedule? Take everything you have been preparing from the other recommendations in this list and put all that information into this document so your (potential) participants can refer to it before and during the sprint. See this primer document from the CarpentryCon@Home 2020 sprint on the Library Carpentry FAIR Data and Software lesson.

There is a number of tools that in combination could be used for running effective curriculum development sprints - we cover a good number of tried and tested technologies under the tools section.

D.2.1 Before The Sprint: Recommendations for GitHub

- **Review open issues on your lesson repository** before the sprint and consider flagging (labelling) existing issues and opening new ones to describe tasks that could be tackled during the event. GitHub Projects can help with this, allowing you to label particular issues within scope for the sprint and even organise them further into categories to help contributors find something to work on. Even if you cannot invest time into managing issues to this extent, make sure open issues include all the contextual information necessary for a new or part-time contributor to understand what needs doing and whether they can help.

D.3 During the Sprint

- **Enable access to documents for shared note-taking.** When possible store all the shareable documents in a central repository or shared drive so they are easy to find. One main document should be created to share links to all the resources that will be used for the sprint. This document should be kept visible to all participants and persist beyond the end of the sprint.

- **Run a "sprint primer" session.** A primer session helps contributors get oriented with the structure of your lesson and its associated repository, the setup required to work on the lesson (e.g. local builds to preview lesson webpages), and the communication channels that will be used during and in-between the sprint sessions. You could run this primer session a week or two days before the sprint begins. This can also be the first session of the sprint event before co-working sessions begin.

- **Start the video conference call 10 mins ahead of official start time.** [**Online**] To allow newcomers and first-time participants to get familiar with tools, test audio and video set up, warm up, etc.

- **Record information about participants and contributions.**

  - Start your event by asking people to sign-in to the shared document by providing names, ORCID identifiers, GitHub usernames, and any other identifiers/handles that can be used to acknowledge their contributions. You can also ask you the participant to optionally provide their pronouns to ensure that everyone is addressed with their preferred pronouns.
  - When working on an online repository, you can use issues to describe and assign tasks. This can also be directly done on the shared notes that everyone has access to. This helps to avoid duplication of effort during the sprint, and gives participants a way of coordinating

    additional discussions among themselves where appropriate to make progress on a task.
  - Prompt participants to add their details to the shared notes, so you can refer to it later and ensure everyone's contributions are recorded and proper credit is given when publishing the lesson.

- **Divide co-working sessions with scheduled check-ins** Plan your sprint to have several co-working sessions that are followed by check-ins or report outs to allow contributors to reflect on their progress. This will also allow them to share what they've achieved, what's blocking their progress, where they would like some feedback and what they intend to do next. We recommend using a web-based shared clock such as cuckoo.team to keep track of co-working, share-out and breaks.

- **Record the check-ins.** If technically possible, and provided all participants consent, record these check-ins (video or written) so that those who could not join can catch up on the latest progress when they (re-)join the sprint. This is particularly important for sprints taking place across multiple sessions/days and/or a wide range of time zones. These check-in discussions can end by taking extensive notes in the shared document.

  - **Assign roles for check-ins.** To ensure everyone has an equal chance to contribute to the discussion and the sprint, the lead organiser(s) should assign meeting roles for the check-in calls. To further promote equity of participation, these roles should be rotated for each day of the sprint, or each separate sprint event. Roles may include:
    * a *Facilitator* who coordinates the discussion and keeps it moving, watching out for raised hands and making sure those with something to contribute get the chance to do so;
    * a *Timekeeper* who monitors the time remaining for the check-in and ensures that all topics are covered in a timely fashion. The Timekeeper prompts the Facilitator/speaker to move on when necessary to cover everything.
    * an *Issue/Task Creator*, who keeps track of actions that arise from the check-in discussion and creates an issue or describes the task in the shared notes.
    * a *Notetaker*, who takes minutes of what is discussed in the check-in. In some instances, it may be possible to combine the roles of Notetaker and Task Creator.

- **Consider participants' roles and expertise.**

  - Consider the power dynamics in the group, e.g. if you have PIs and early-career PhD candidates participating, have seperate breakout rooms for each of those groups so participants can comfortably discuss with peers (but report out at the end so folks can learn from other perspectives), have a moderator who keeps track how much folks are speaking and provide space and (gentle) prompts to encourage the lesser-spoken folks to speak, etc

- **Use a chat channel for communication outside check-ins.** To allow participants to “unplug” from video calls/move to separate locations for the bulk of the sprint, use a text-based chat such as Slack or Gitter for discussion when not checking in. If your chosen platform allows it, minimise global notifications to sprint participants by making use of threaded conversations. This will allow contributors to focus on their chosen task unless involved in a conversation directly relevant to them.

- **Prepare breakout rooms for small group discussion and co-working between check-ins.** [**Online**] Open a small number of break out rooms for sprint participants to use during working sessions. This allows the participants to self-organize if they need to discuss something they are working on or co-work on a particular task. In Zoom, select the option to “Let participants choose room”. For [**in-person**] sprints, prepare an equivalent set of different rooms/locations for participants to gather and discuss in smaller groups without fear of disturbing the whole team.

- **Provide a way for contributors to mark their work ready for review.** If you prefer to require material is reviewed before being included in the lesson, give participants a way to signal when their contributions are ready. You might consider providing guidance/assigning responsibility for reviewers of particular sections/types of contribution. If your lesson is being developed on GitHub, see the section below for more specific recommendations.

- **Reviewing contributions.** Allocate 1-2 sessions in the sprint to review and approve contributions made by others. If this is not planned in the event, sprints may end with an influx of new material as contributors finish up what they were working on as the event closed. Make sure that each contribution is reviewed and if possible (and where appropriate), merged into the lesson. This allows contributors to leave the event with a sense of accomplishment from the sprint, and ties up any loose ends before new tasks are taken on to develop the material further. If the participants who were originally assigned to review have had to leave before the review session, the organisers should delegate the review task to those who are available and willing to help finish the review process on time.

- **Ask for feedback.** Allocate a short feedback session before closing the event. Feedback should be kept short and recorded anonymously via a feedback form or shared document (that does not require signing in). Ask your participants to reflect on the event in terms of what worked and what can be improved.

D.3.1 During the Sprint: Recommendations for GitHub

- **Consider running a skill-up session for contributors.** Depending on their previous experience with the platform, contributors may benefit from learning/being reminded how to use GitHub, working on branches/forks, and the associated tools on their local system. Providing a session/resources on this will also make your sprint less intimidating for newcomers to lesson development, which may increase participation. GitPod may offer a way for participants to contribute to the lesson without the need for setting up a local development environment. As with the "primer" session mentioned above, this skill-up may fit best at the beginning of the sprint. This can also run as a separate event that takes place in the days leading up to the full sprint.

- **Flag pull requests as a draft and ready for review as you work.** Opening draft pull requests early, while work on a task is ongoing, can help contributors understand where progress is being made and identify tasks/issues that still need to be tackled. Marking pull requests as ready for review helps maintainers prioritise their focus during the sprint and maintains momentum by ensuring that completed work is merged into the lesson as soon as possible. Opening draft pull requests is particularly important at the end of a sprint, to capture progress up to that point even if a task hasn't yet been completed.

- **Plan who will review pull requests.** Make a plan for who will review each pull request (PR). As a group, you can create a list of reviewers by subject so the person who opens the pull request knows who is interested and capable of reviewing and can request a review accordingly. Alternatively, you may decide on a circle of reviewers. E.g. Aaliyah reviews Juan's PRs, Juan reviews Alex's PRs, and Alex reviews Aaliyah's PRs. Another option might be to ask for a reviewer when planning who will do the task. For particularly small groups where reviewing is more likely to become a bottleneck, or in cases where the main objective of the sprint is to create quantity rather than quality, you may decide it is acceptable for contributors to merge their own pull requests.

- **Link pull requests to open issues.** Mentioning open issues in pull requests (even when a work in progress) informs people about which issues are being worked on. As mentioned in the point above, this is particularly useful to ensure progress can continue on the lesson after the sprint has finished.

- **Record contributions that do not (directly) result in commits.** Some participants may be more comfortable writing material on a different platform e.g. Google Docs, which is later transferred to GitHub by someone else. Make use of GitHub's commit co-author feature where possible or, if the author does not have a GitHub account, make sure their name and other details are included in your README, AUTHORS, or any other listing/metadata about contributors to the project (`.zenodo.json`, `.allcontributorsrc`, etc).

- **Review open pull requests.** At the end of the sprint, organisers should ensure that any pull requests marked as ready for review are processed and (where appropriate) merged, so that these contributions are not forgotten or left to go stale before changes can be requested.

D.4 After the Sprint

- **Celebrate the progress.** Though lesson development is an iterative process, every contribution counts towards the completion of a lesson. It is, therefore, important to celebrate the progress you and your colleagues make at the lesson development sprint. Take a break (if you can) before meeting the organising committee for a debrief call.
- **List completed and outstanding tasks.** Create GitHub issues or share a document to highlight the status of different tasks. Set a deadline for the completion of outstanding tasks to share online with all the contributors. These will help you and other contributors understand the next steps when working together on the lesson as well as continue working on the lesson asynchronously.
- **Plan the next steps!** Review the shared documents from the sprint and highlight successes and place for improvements to include in a report. Feedback and reflections from running a sprint often inform a new set of changes and updates to be made in the event format or lesson material, discuss and document those for planning the next events. Plan to host a periodic co-working call to keep collaborators engaged or schedule the next sprint to ensure that momentum is maintained on the lesson.
- **Write up a report and share it with your contributors.** A summary of progress made and next steps (with links to open issues/work-in-progress pull requests) will help you and your contributors appreciate the progress made during your sprint, and encourage efforts on the lesson to continue. Consider turning this overview into a blog post to share your experience and achievements with the wider community!
- **Contribute to these recommendations.** While organising and running your sprint, did you find any useful resources or steps not already covered in this guide? Do you have any advice or guidance for others who may follow these recommendations in future? If so, please contribute back to improve this resource! To tell us about changes you would like to see to these recommendations, please email the authors (`tobyhodges@carpentries.org`), open an issue or contribute your changes directly in a new pull request.

D.4.1 After the Sprint: Recommendations for GitHub

- **Capture outstanding tasks in new issues on the lesson repository.** You may also wish to assign these to specific contributors for follow up after the sprint.

- **Label issues to encourage asynchronous contributions.** Issue labels, such as "help-wanted" and "good-first-issue," provide valuable information to potential contributors about where their contributions could be most valuable and how they could begin to get involved with developing your lesson. Issues labelled "good first issue" will appear automatically on the `/contribute` page of your repository, providing a quick overview of ways to get involved that have a low barrier for entry. Other labels can be used in a similar way to automatically display listings of issues to potential contributors. For example, The Carpentries publishes a Help Wanted page that provides their community with an overview of open issues with the "help-wanted" label on many of their repositories.

- **Clean up branches on your repository.** Sprints often leave behind a large number of merged branches that can be removed from the lesson repository. Minimising the number of "stale" branches reduces clutter on your repository, allows contributors to name their branches without fear of an unintentional clash and helps people working with local branches.

D.5 Tools and Resources

- Cuckoo is a **shared countdown timer** for collaborative co-working sessions. Create a page and share the URL with your contributors; everyone connected to the page will be notified when the set countdown ends. Shared timers such as this can be useful to help coordinate sprint sessions between check-ins.
- GitPod provides **a cloud-based development environment** for projects. The service can deliver an interactive development environment for a project on GitHub, with dependencies and environment setup already taken care of, allowing contributors to begin working on a project without needing to deal with this overhead on their local machine. The GitPod documentation provides instructions to configure your project to use the service.
- **Shared note-taking** tools:
  - Etherpad provides a relatively simple shared note-taking interface. Useful features include colouring to differentiate contributions from individual participants, an integrated chat window, and line numbering. Open Etherpad services include FIXME

  - HedgeDoc (formerly CodiMD) provides an interface for collaborative writing in Markdown. Useful features include a split-panel view for simultaneous editing and previewing, built-in support for concept mapping with GraphViz and Mermaid.js, and line numbering. The Carpentries provides a CodiMD instance for use in activities and events relating to their community, and HackMD and Cryptpad provide similar, free services to a general audience.
  - Google Docs provides an interface for collaborative writing similar word processing software such as Microsoft Word. Useful features include threaded comments, a mode for suggesting changes instead of directly editing the document, and visual formatting without the need for text markup.

- timeanddate.com provides many **services related to conversion between time zones**. Useful features include a meeting planner suggesting suitable times for events with attendees in multiple regions, an event announcement system displaying the start time and date of an event converted to the visitor's locale, and a time zone converter to check what time it is right now in another location around the world.
- Google Calendar and other **timezone-aware calendar applications** can be used to create entries for your event that will automatically display in the local time of the person viewing the page/entry. Alongside listing event times in UTC with a link to timeanddate.com (see above), timezone-aware calendar events are essential to make your sprint inclusive to an international community.
- The Carpentries Curriculum Development Handbook provides **an overview of the backwards design approach** recommended for lesson development by The Carpentries. It includes guidance on identifying the target audience of your lesson, defining meaningful learning objectives, designing appropriate examples and exercises, and the various stages of the lesson life cycle.
- Slack provides a workplace **messaging platform** for threaded text discussion. Useful features include multiple open and private channels, document and image sharing, and a wide range of plugins to interface with other systems. Some features require a paid subscription. Members of The Carpentries community can create a channel for discussion around development of their lesson in the organisation's Slack workspace
- Gitter is another **instant messaging service**. It lacks some of the features of Slack but does not require a paid plan to remove limits on usage and it integrates well with GitHub, e.g. by allowing references to particular repositories, issues, and pull requests.

These and many other tools that could be useful for organising and running your sprint are also described in the Tools for Remote Collaboration chapter of The Turing Way.

D.6 Acknowledgments

These recommendations were originally developed in a separate GitHub repository by Steve Crouch, Toby Hodges, Zhian Kamvar, Aleksandra Nenadic, Malvika Sharan, Sarah Stevens, and Emmy Tsang. Ideas, suggestions, and guidance were provided by Sarah Alidoost, Chris Erdmann, Kristina Hettne, and Liz Stokes.

D.7 Further Reading

- The Guide for Collaboration section of The Turing Way has many more recommendations and guides to help your team collaborate effectively. Many other sections of the book are also relevant.
- The Turing Way's process for organising and running a Book Dash has many similarities to the recommendations made here for lesson sprints. We recommend that you explore this resource, which also includes checklist and schedule templates that could fit a lesson sprint with some minor adjustments.
- In addition to the rest of this Handbook, Greg Wilson's Teaching Tech Together provides a more in-depth overview of the reverse design approach to lesson development as well as many useful pointers for teaching and collaborating.
- The Carpentries Instructor Training curriculum is another valuable source of advice for writing and teaching lessons.

Bibliography

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., and Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.

Jordan, K., Michonneau, F., and Weaver, B. (2018). Analysis of Software and Data Carpentry's Pre- and Post-Workshop Surveys.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6):1–20.